SYBASE®
An SAP® Company

ASE Transact-SQL® ユーザーズ・ガイド

Adaptive Server® Enterprise

15.7

ドキュメント ID：DC36429-01-1570-01
改訂：2011 年 9 月

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

目次

第 1 章

SQL ビルディング・ブロック

Adaptive Server での SQL

1970 年代後半に IBM の San Jose Research Laboratory によって最初に開発されて以来、構造化問合せ言語 (SQL: Structured Query Language) は、多くのリレーショナル・データベース管理システムに採択され、適合されてきました。SQL は米国規格協会 (ANSI: American National Standards Institute) および国際標準化機構 (ISO: International Organization for Standardization) によって、公式のリレーショナル問い合わせ言語として承認されています。

Transact-SQL® は Sybase® の SQL 拡張であり、IBM SQL およびその他ほとんどの商用実装の SQL と互換性があります。サマリ計算、ストアド・プロシージャ (定義済み SQL 文)、エラー処理などの追加機能が備わっています。

SQL にはデータベースを問い合わせる (データベースからデータを検索する) ためのコマンドおよび新しいデータベースと「データベース・オブジェクト」の作成、新しいデータの追加、既存データの修正、その他の機能のためのコマンドも含まれています。

注意 使用する Adaptive Server で Java が実行可能である場合は、データベースに Java クラスをインストールして使用できます。標準の Transact-SQL コマンドを使用して、Java 演算を呼び出して Java クラスを格納できます。『Adaptive Server Enterprise における Java』を参照してください。

Adaptive Server には、pubs2 と pubs3 のサンプル・データベースが用意してあります。これらは Adaptive Server のマニュアルに記載されている例の大半で使用されています。「付録 A pubs2 データベース」と「付録 B pubs3 データベース」を参照してください。

クエリ、データ修正、およびコマンド

SQL の「クエリ」は、select コマンドを使用してデータを要求します。たとえば、次に示すクエリは、カリフォルニア州に住む作家を要求します。

```
select au_lname, city, state
from authors
where state = "CA"
```

「データ修正」とは、データの追加、削除、変更のことで、それぞれ insert コマンド、delete コマンド、update コマンドを使用して行います。次に例を示します。

```
insert into authors (au_lname, au_fname, au_id)
values ("Smith", "Gabriella", "999-03-2346")
```

その他の SQL コマンドは、テーブルの削除やユーザの追加などの管理オペレーションを実行するための命令です。次に例を示します。

```
drop table authors
```

それぞれのコマンドや SQL 文は、insert など、実行する基本の演算を示す「キーワード」で始まります。SQL コマンドの多くには、1 つ以上の「キーワード・フレーズ」または「句」があり、特定の必要性に合わせてコマンドに追加します。クエリを実行すると、Transact-SQL は結果を表示します。クエリ内で指定した基準を満たすデータがない場合、その旨を示すメッセージが表示されます。データ修正文および管理文はデータ検索を行わないので、結果を表示しません。Transact-SQL は、データ修正や他のコマンドが実行されたかどうかをユーザに知らせるメッセージを表示します。

テーブル、カラム、およびロー

リレーショナル・データベース管理システムでは、ユーザはテーブルに格納されたデータに対してアクセスおよび修正を行います。SQL は、特にリレーショナル・モデルのデータベース管理用に設計されています。

テーブルの各ローまたはレコードは、人間、会社、売り上げなどのデータを 1 組として記述します。各カラムまたはフィールドは、人間の名前や住所、会社の名前や社長、売り上げの量など、データの特性を記述します。

リレーショナル・データベースは、相互に関連付けが可能な一連のテーブルで構成されています。通常は、データベースには多数のテーブルが含まれます。

関係演算

リレーショナル・システムにおける基本のクエリ演算は、選択 (制約とも呼ばれる)、射影、ジョインです。これはすべて SQL の select コマンド内で結合できます。

「選択」は、テーブルのローのサブセットです。select クエリには制限条件を指定します。たとえば、カリフォルニア州に住む作家すべてのローのみを参照するには、次のように入力します。

```
select *
from authors
where state = "CA"
```

「射影」は、テーブルのカラムのサブセットです。たとえば、このクエリは、作家の住所や電話番号その他の情報は表示せずに、名前と都市だけを表示できます。

```
select au_fname, au_lname, city
from authors
```

「ジョイン」は、特定のフィールドの値を比較して、複数のテーブルにあるローをリンクします。たとえば、au_id (作家の ID 番号) カラムと au_lname (作家の姓) カラムを含む作家の情報を格納するテーブルが 1 つあるとします。もう 1 つのテーブルには、作家の ID 番号を指定するカラム (au_id) を含む、本のタイトル情報が格納されているとします。それぞれのテーブルの au_id カラムの値の等価性をテストして、authors テーブルと titles テーブルをジョインできます。一致があるたびに、2 つのテーブルからのカラムを格納する新しいローが 1 つ作成されて、ジョインの結果の一部として表示されます。ジョインは、選択された一致するローの選択されたカラムだけが表示されるように、射影および選択と結合されることがよくあります。

```
select *
from authors, publishers
where authors.city = publishers.city
```

コンパイル済みオブジェクト

Adaptive Server は、各データベースに関する重要な情報を取得したり、ユーザによるデータのアクセスや操作をサポートしたりするために、「コンパイル済みオブジェクト」を使用します。コンパイル済みオブジェクトは、sysprocedures テーブルにあるエントリを必要とする任意のオブジェクトです。これには、次のものがあります。

- 検査制約
- デフォルト
- ルール
- ストアド・プロシージャ

- 拡張ストアド・プロシージャ
- トリガ
- ビュー
- 関数
- 計算カラム
- 分割条件

コンパイル済みオブジェクトは、コンパイル済みオブジェクトを記述および定義する SQL 文である「ソース・テキスト」から作成されます。コンパイル済みオブジェクトが作成されると、Adaptive Server は次の処理を行います。

1 構文エラーを検出してソース・テキストを解析し、解析ツリーを生成します。

2 解析ツリーを正規化して、ユーザ文をバイナリ・ツリー・フォーマットで表す正規化ツリーを作成します。これはコンパイル済みオブジェクトです。

3 コンパイル済みオブジェクトを sysprocedures テーブルに格納します。

4 ソース・テキストを syscomments テーブルに格納します。

ソース・テキストの保存またはリストア

コンパイル済みオブジェクトに、syscomments テーブル内のソース・テキストと一致するものがない場合は、次に示すいずれかの手順で、syscomments にソース・テキストをリストアできます。

- バックアップからソース・テキストをロードする。
- ソース・テキストを手作業で再作成する。
- コンパイル済みオブジェクトを作成したアプリケーションを再インストールする。

ソース・テキストの検証および暗号化

Adaptive Server では、ソース・テキストの存在を確認し、必要に応じてそのテキストを暗号化できます。ソース・テキストの操作には、次のコマンドを使用します。

- sp_checkresource － コンパイル済みオブジェクトごとに、ソース・テキストが syscomments にあることを検証します。
- sp_hidetext － syscomments テーブル内のコンパイル済みオブジェクトのソース・テキストを暗号化します。
- sp_helptext － ソース・テキストが syscomments にある場合は、それを表示します。ない場合は、ソース・テキストがないことを通知します。
- dbcc checkcatalog － ソース・テキストがないことを通知します。

ANSI 標準への準拠

SQL の標準規格によって定義されている動作の中には、Adaptive Server のアプリケーションと互換性のないものがあります。Transact-SQL は、このような動作の設定と解除 (オン/オフ) を可能にする set オプションを提供します。

準拠動作は、デフォルトですべての Embedded SQL™ プリコンパイラ・アプリケーションに対して有効になります。SQL 標準動作に対応する必要のある他のアプリケーションは、表 1-1 の初級レベル ANSI SQL のオプションを使用できます。『リファレンス・マニュアル：コマンド』を参照してください。

表 1-1: 初級レベル ANSI SQL に準拠するためのコマンド・フラグ設定

オプション	設定値
ansi_permissions	on
ansinull	on
arithabort	off
arithabort numeric_truncation	on
arithignore	off
chained	on
close on endtran	on
fipsflagger	on
quoted_identifier	on
string_rtruncation	on
transaction isolation level	3

以降の項では、標準の動作とデフォルトの Transact-SQL 動作との違いを説明します。

連邦情報処理規格 (*FIPS*) フラグ

ANSI SQL 規格に準拠する必要があるアプリケーションを作成するユーザには、Adaptive Server は set fipsflagger オプションを提供します。このオプションが on になっていると、初級レベルの ANSI SQL では許可されていない Transact-SQL 拡張機能を含むすべてのコマンドは情報メッセージを生成します。このオプションは拡張機能を無効にするのではありません。ANSI SQL 以外のコマンドを発行すると、処理は完了します。

FIPS については、『システム管理ガイド第 1 巻』の「第 12 章セキュリティの概要」を参照してください。

連鎖トランザクションと独立性レベル

Adaptive Server は、オプションで SQL 規格準拠の「連鎖」トランザクション動作を備えています。連鎖モードでは、すべてのデータ検索コマンドおよびデータ修正コマンド (delete、insert、open、fetch、select、update) は暗黙的に「トランザクション」を開始します。このような動作は多くの Transact-SQL アプリケーションと互換性がないため、Transact-SQL スタイル (つまり、「非連鎖」) のトランザクションがデフォルトになります。

連鎖トランザクション・モードは set chained オプションで開始できます。set transaction isolation level オプションは、トランザクションの独立性レベルを制御します。「第 23 章 トランザクション：データの一貫性およびリカバリ」を参照してください。

識別子

初級レベル ANSI SQL に準拠するには、識別子は次のようであってはなりません。

- シャープ記号 (#) で開始する
- 18 文字を超えている
- 小文字を含んでいる

Adaptive Server は、テーブル、ビュー、およびカラム名の区切り識別子をサポートしています。区切り識別子は二重引用符で囲まれたオブジェクト名です。これを使用することで、オブジェクト名の制限をある程度回避できます。

区切り識別子を認識するには、set quoted_identifier オプションを使用します。このオプションが on になっていると、二重引用符で囲まれたすべての文字は識別子として扱われます。この動作は既存の多くのアプリケーションと互換性がないため、このオプションのデフォルト設定は off になっています。

SQL 標準スタイルのコメント

Transact-SQL では、コメントは “/*” と “*/” で区切られ、ネストできます。Transact-SQL は SQL 規格スタイルのコメントもサポートしています。これはマイナス記号を 2 つ並べた後に、コメントが続き、改行で終わる任意の文字列です。

```
select "hello" -- this is a comment
```

Transact-SQL の “/*” と “*/” コメント・デリミタは完全にサポートされていますが、Transact-SQL コメント内での “--” は認識されません。

文字列の右側トランケーション

string_rtruncation set オプションは、SQL 規格との互換性のための文字列の暗黙的なトランケーションを制御します。暗黙のトランケーションを行わないようにして SQL 規格の動作を実行するには、このオプションを有効にします。

update 文および *delete* 文に必要なパーミッション

ansi_permissions set オプションは、delete 文および update 文に、どのパーミッションが必要かを決定します。このオプションが有効の場合、Adaptive Server はこれらの文に対して ANSI SQL のより厳重なパーミッション要件を使用します。デフォルトでは、この動作は既存の多くのアプリケーションと互換性がないため、このオプションは無効に設定されています。

算術エラー

set の arithabort および arithignore オプションは、次のようにして ANSI SQL 規格に準拠します。

- arithabort arith_overflow は、0 による除算エラーや精度の消失の後の動作を指定します。デフォルト設定の arithabort arith_overflow on では、エラーが発生したトランザクション全体がロールバックされます。arithabort arith_overflow on が設定されていると、トランザクションを含まないバッチでエラーが発生した場合、バッチ内のエラーより前のコマンドはロールバックしません。ただし Adaptive Server は、エラーを起こした文に続くバッチ内の文は実行しません。

  arithabort arith_overflow off を設定した場合には、Adaptive Server はエラーを発生させた文をアボートしますが、トランザクションまたはバッチ内の残りの文の処理を継続します。

- arithabort numeric_truncation は、真数値型による位取りの消失の後の動作を指定します。デフォルト設定の on では、エラーを発生した文をアボートしますが、トランザクションまたはバッチ内の他の文の処理を続けます。arithabort numeric_truncation off を設定した場合、Adaptive Server はクエリ結果をトランケートして処理を継続します。ANSI SQL 規格に準拠するには、set arithabort numeric_truncation on を入力します。

- arithignore arith_overflow は、0 による除算エラーや精度の消失の後に Adaptive Server がメッセージを表示するかどうかを指定します。デフォルト設定の off では、このようなエラーの後に警告メッセージを表示します。arithignore arith_overflow on を設定すると、これらのエラーが発生しても警告メッセージは表示されません。ANSI SQL 規格に準拠するには、set arithignore off を入力します。

注意 JDBC コードの警告の処理については、『jConnect for JDBC プログラマーズ・リファレンス』を参照してください。「プログラミング情報」の章の「エラー・メッセージの処理」の項にある「警告として返される数値エラーの処理」を参照してください。

同義のキーワード

表 1-2 に SQL 規格の互換性のために追加され、既存の Transact-SQL キーワードと同義のキーワードを示します。

表 1-2: ANSI と互換性のあるキーワードの同義語

現在の構文	追加の構文
commit tran、commit transaction、rollback tran、rollback transaction	commit work、rollback work
any	some
grant all	grant all privileges
revoke all	revoke all privileges
max (*expression*)	max ([all | distinct]) *expression*
min (*expression*)	min ([all | distinct]) *expression*
user_name *function*	user *keyword*

null の扱い

set オプション ansinull は、SQL 等号 (=) または不等号 (!=) 比較および集合関数における null 値のオペランドの評価が SQL 標準と互換性があるかどうかを判別します。このオプションは、create table などの他の SQL 文で Adaptive Server が null 値をどのように評価するかには影響しません。

命名規則

Adaptive Server によって認識される文字は、インストール言語やデフォルトの文字セットによって、ある程度制限されます。したがって、SQL 文およびサーバ内のデータに使用できる文字はインストールごとに異なり、デフォルト文字セット内の定義によって、ある程度決定されます。

SQL 文は厳密な構文規則および構造規則に従わなくてはならず、演算子、定数、SQL キーワード、特殊文字、および「識別子」を含むことができます。識別子はデータベース名やテーブル名など、サーバ内のデータベース・オブジェクトです。命名規則は SQL 文の各部で異なります。演算子、定数、SQL キーワード、および Transact-SQL 拡張子は、従うべき命名規則が識別子よりも厳密で、演算子および特殊文字を含むことができません。しかし、サーバ内に格納されるデータである識別子の命名は、次に示すようにより緩い規則になっています。

以降の項では、文の各部に使用される文字のセットについて説明します。識別子の項では、データベース・オブジェクトの命名規則についても説明します。

SQL データ文字

SQL データ文字のセットは大規模なセットで、SQL 言語文字と識別子文字はいずれもこのセットから取得します。Adaptive Server 文字セットにある文字は、シングルバイトとマルチバイトを含めていずれもデータ値に使用できます。

SQL 言語文字

SQL キーワード、Transact-SQL 拡張子、**比較演算子**、> と < などの特殊文字は、7 ビット ASCII 値 A 〜 Z、a 〜 z、0 〜 9 に加え、次に示す ASCII 文字だけで表すことができます。

表 1-3: SQL で使用される ASCII 文字

文字	説明
;	(セミコロン)
(	(左カッコ)
)	(右カッコ)
,	(カンマ)
:	(コロン)
%	(パーセント記号)
-	(マイナス記号)
?	(疑問符)
’	(一重引用符)
"	(二重引用符)
+	(プラス記号)
_	(アンダースコア)
*	(アスタリスク)
/	(スラッシュ)
	(スペース)
<	(演算子、より小さい)
>	(演算子、より大きい)
=	(等号演算子)
&	(アンパサンド)
|	(垂直バー)
^	(曲折アクセント)
[	(左角カッコ)
]	(右角カッコ)
@	(at 記号)
~	(波型記号)
!	(感嘆符)
$	(ドル記号)
#	(シャープ記号)
.	(ピリオド)

識別子

データベース・オブジェクトの命名規則は、Adaptive Server ソフトウェアおよびマニュアル全体を通して適用されます。ほとんどのユーザ定義識別子の最大長は 255 バイトであり、それ以外の識別子の最大長は 30 バイトです。いずれの場合でも、バイト制限は、マルチバイト文字が使用されているかどうかに関係しません。表 1-4 は、さまざまな識別子のバイト制限を示しています。

表 1-4: 識別子のバイト制限

255 バイトに制限される識別子	30 バイトに制限される識別子
テーブル名	カーソル名
カラム名	サーバ名
インデックス名	ホスト名
ビュー名	ログイン名
ユーザ定義データ型	パスワード
トリガ名	ホスト・プロセス ID
デフォルト名	アプリケーション名
ルール名	初期言語名
制約名	文字セット名
ストアド•プロシージャ名	ユーザ名
変数名	グループ名
JAR 名	データベース名
ライトウェイト・プロセス (LWP) または動的文の名前	キャッシュ名
関数名	論理デバイス名
時間範囲の名前	セグメント名
関数名	セッション名
アプリケーション・コンテキスト名	実行クラス名
	エンジン名
	静止タグ名

識別子の最初の文字は、Adaptive Server で使用している文字セット定義において、アルファベット文字であると宣言されている必要があります。@ 記号や _ (アンダースコア文字) も使用できます。識別子の最初に @ 記号がある場合は、「ローカル変数」を示します。

テンポラリ・テーブルが tempdb の外部で作成された場合、テンポラリ・テーブル名は #(シャープ記号) で開始する必要があります。それ以外の場合は、“tempdb” で開始されます。テンポラリ・テーブルを作成する場合、名前が 238 バイトを超えないようにしてください。それは、テーブル名をユニークにするために、Adaptive Server が 17 バイトのサフィックスを付加するためです。238 バイトを超える名前を使用してテンポラリ・テーブルを作成すると、Adaptive Server は、最初の 238 バイトだけを使用して、それに 17 バイトのサフィックスを付加します。

識別子の 2 文字目以降には、アルファベット、数値、または $、#、@、_、¥ (円)、£ (通貨ポンド) などの文字として宣言されている文字を含むことができます。ただし、“@@myobject” などのように、名前付きオブジェクトの前に @@ 記号を 2 つ並べて使用することはできません。この命名規則は、Adaptive Server が自動的に更新するシステム定義変数である「グローバル変数」用に予約されています。

Adaptive Server の大文字と小文字の区別は、サーバがインストールされたときに設定され、システム管理者のみが変更可能です。使用しているサーバの設定を確認するには、次のコマンドを実行します。

```
sp_helpsort
```

大文字と小文字を区別しないサーバでは、識別子 MYOBJECT、myobject、MyObject (および大文字と小文字のすべての組み合わせ) は、同じであると認識されます。これら 3 つのオブジェクトのうち 1 つしか作成できませんが、どの大文字と小文字の組み合わせを使用してもかまいません。

識別子には、埋め込みスペースおよび SQL 予約語を使用できません。valid_name を使用して、作成した識別子が Adaptive Server に受け入れ可能かどうかを調べることができます。

```
select valid_name ("@name", 255)
```

『リファレンス・マニュアル：ビルディング・ブロック』の「第 5 章 予約語」と「第 2 章 Transact-SQL 関数」を参照してください。

マルチバイト文字セット

マルチバイト文字セットでは、より広い範囲の文字を識別子に使用できます。たとえば、日本語がインストールされているサーバでは、全角または半角カタカナ、ひらがな、漢字、ローマ字、キリル文字、ギリシャ文字、ASCII を、識別子の 1 文字目に使用できます。

半角カタカナは日本語システムの識別子では有効ですが、異機種間システムでの使用はおすすめできません。半角カタカナは EUC-JIS 文字セットと Shift-JIS 文字セットの間では変換できません。

これは 8 ビット欧州文字のいくつかにもあてはまります。たとえば OE 合字の “Œ” (コード・ポイント 0xCE) は、Macintosh 文字セットの一部ですが、ISO 8859-1 (iso_1) 文字セットには存在しません。Macintosh から ISO 8859-1 文字セットに変換中のデータに “Œ” がある場合、変換エラーが発生します。

変換できない文字がオブジェクト識別子に含まれていると、クライアントはそのオブジェクトに直接アクセスできなくなります。

区切り識別子

「区切り識別子」は、二重引用符で囲まれたオブジェクト名です。区切り識別子を使用することによって、オブジェクト名に関する一定の制限を回避できます。二重引用符は、テーブル、ビュー、およびカラムの名前を区切るのに使用できます。それ以外のデータベース・オブジェクトには使用できません。

区切り識別子は予約語にすることができ、アルファベット以外の文字で開始可能で、他では使用できない文字を含むことができます。253 バイトを超えることはできません。シャープ記号 (#) を引用符付き識別子の最初に使用することはできません(この制限は Adaptive Server 11.5 以降のバージョンに適用されます)。

区切り識別子を作成または参照する前に、次のコマンドを実行してください。

```
set quoted_identifier on
```

これによって、Adaptive Server は区切り識別子を認識するようになります。文中で引用符付き識別子を使用するときは、そのたびに二重引用符で囲んでください。次に例を示します。

```
create table "1one"(col1 char(3))
select * from "1one"
create table "include spaces" (col1 int)
```

注意 区切り識別子は bcp とともに使用することはできません。これらの識別子は、どのフロントエンド製品でもサポートされておらず、システム・プロシージャとともに使用すると、予期しない結果が起こることがあるためです。

---

quoted_identifier オプションが on に設定されている間は、一重引用符で文字やデータ文字列を囲んでください。これらの文字列を二重引用符で区切ると、Adaptive Server はこれを識別子として扱います。たとえば *1onetable* の *col1* に文字列を挿入するには、次のように指定します。

```
insert "1one"(col1) values ('abc')
```

次のように入力しないでください。

```
insert "1one"(col1) values ("abc")
```

カラムに一重引用符を挿入するには、連続する 2 つの一重引用符を使用してください。たとえば、col1 に “a'b” を挿入するには、次のようにします。

```
insert "1one"(col1) values('a''b')
```

引用符を含む構文

あるセッションで quoted_identifier オプションを on に設定するときは、構文エラーを引き起こす可能性のあるオブジェクト名を二重引用符で区切ります。文字列は一重引用符で囲んでください。あるセッションで quoted_identifier オプションを off (デフォルト値) に設定するときは、文字列を二重引用符または一重引用符で区切ります (識別子は引用できません)。

この例では、テーブル 1one を作成します。名前が数字で始まっており、識別子のルールから外れているので、引用符で囲む必要があります。

```
set quoted identifier on
go
create table "1one" (c1 int)
```

create table およびその他のほとんどの SQL 文では、テーブルやその他の SQL オブジェクトを指定するために識別子が必要とされますが、一部のコマンドや関数などでは、quoted_identifier オプションが on にされているかどうかにかかわらず、オブジェクト名が文字列として提示される必要があります。次に例を示します。

```
select object_id('1one')
-----------------------
                  896003192
```

引用符を 2 つ使用すると、引用符付き識別子に埋め込み二重引用符を含めることができます。これにより、embedded”quote という名前のテーブルが作成されます。

```
create table "embedded""quote" (c1 int)
```

ただし、文の構文でオブジェクト名を文字列として表現することが要求されている場合には、引用符を 2 つ使用する必要はありません。

```
select object_id('embedded"quote')
```

角カッコ区切り識別子

Sybase は、角カッコ識別子もサポートしています。角カッコ識別子の動作は、この識別子を使用するために quoted_identifier オプションを on に設定する必要がない点を除けば、引用符付き識別子と同じです。

```
create table [bracketed identifier](c1 int)
```

区切り識別子のある角カッコをサポートすることにより、プラットフォームの互換性が向上します。

一意性と修飾の規則

データベース・オブジェクトの名前は、データベース内でユニークである必要はありません。しかし、カラム名とインデックス名はテーブル内でユニークである必要があり、その他のオブジェクト名は、データベース内の所有者ごとにユニークである必要があります。Adaptive Server では、データベース名を一意にしてください。

テーブル内でユニークでない名前を使用してカラムを作成したり、テーブル、ビュー、ストアド・プロシージャなど別のデータベース・オブジェクトを、同じデータベース内で既に使用している名前で作成したりすると、Adaptive Server はエラー・メッセージを返します。

テーブルやカラムにそれを修飾する他の名前を追加することによってユニークに識別できます。データベース名、所有者名、カラムの場合はテーブル名またはビュー名を使用してユニーク ID を作成できます。このような修飾子は、ピリオドで 1 つずつ区切られます。

たとえば、ユーザ “sharon” が pubs2 データベース内の authors テーブルを所有している場合、そのテーブルの city カラムのユニークな識別子は、次のようになります。

```
pubs2.sharon.authors.city
```

その他のデータベース・オブジェクトにも、同じ命名構文が当てはまります。同様の方法で、任意のオブジェクトを参照できます。

```
pubs2.dbo.titleview
```

```
dbo.postalcoderule
```

set コマンドの quoted_identifier オプションが on の場合、修飾されたオブジェクト名の各部分を二重引用符で囲むことができます。引用符が必要なそれぞれの修飾子について、引用符を一組ずつ使います。たとえば、次のようにします。

```
database.owner."table_name"."column_name"
```

次のように入力しないでください。

```
database.owner."table_name.column_name"
```

create 文内では、常に完全な命名構文を使用できるわけではありません。現在作業中のデータベース以外のデータベースでは、ビュー、プロシージャ、ルール、デフォルト、またはトリガを作成できないためです。命名規則は、構文中で次のように示されます。

```
[[database.]owner.]object_name
```

または

```
[owner.]object_name
```

owner のデフォルト値は現在のユーザで、*database* のデフォルト値は現在のデータベースです。ユーザが create 文以外の SQL 文中のオブジェクトをデータベース名や所有者名で修飾せずに参照すると、Adaptive Server はまずそのユーザが所有するすべてのオブジェクトを参照してから、「データベース所有者」が所有するオブジェクトを参照します。オブジェクトを識別するのに十分な情報が Adaptive Server に与えられていれば、オブジェクト名の各要素をすべて入力する必要はありません。中間の要素を省略し、その位置をピリオドで示すことができます。

```
database..table_name
```

前述の例では、テーブルを作成するときにこの構文を使用する場合は、最初の要素を含む必要があります。最初の要素を省略すると、..mytable といったテーブルが作成されてしまいます。この命名規則では、カーソル更新などの一定の動作をこのようなテーブルで実行できません。

同じ文中でカラム名とテーブル名を修飾する場合は、それぞれに同じ省略形を使用してください。これは文字列として評価され、一致する必要があります。一致しない場合はエラーが返されます。カラム名に異なるエントリを使用した 2 つの例を示します。2 番目の例は正しくない使い方なので、実行できません。カラム名の構文スタイルが、テーブル名で使用している構文スタイルと一致していないためです。

```
select pubs2.dbo.publishers.city
from pubs2.dbo.publishers

city
-----------------------
Boston
Washington
Berkeley

select pubs2.sa.publishers.city
from pubs2..publishers
```

```
The column prefix "pubs2.sa.publishers" does not match
a table name or alias name used in the query.
```

リモート・サーバ

ユーザはリモート Adaptive Server でストアド・プロシージャを実行できます。ストアド・プロシージャからの結果は、プロシージャを呼び出した端末に表示されます。リモート・サーバとストアド・プロシージャを識別する構文は、次のとおりです。

```
[execute] server.[database].[owner].procedure_name
```

リモート・プロシージャ・コール (RPC) がバッチ内の最初の文であるときは、execute キーワードを省略できます。他の SQL 文が RPC の前にあるときは、execute または exec を使用してください。サーバ名とストアド・プロシージャ名の両方を含める必要があります。データベース名を省略すると、Adaptive Server はデフォルト・データベース内で procedure_name を探します。データベース名を指定する場合、自分がそのプロシージャを所有しているかまたはデータベース所有者が所有しているのでなければ、プロシージャ所有者の名前も指定する必要があります。

次に示す文は、GATEWAY サーバにある pubs2 データベース内のストアド・プロシージャ byroyalty を実行します。

文	注意
`GATEWAY.pubs2.dbo.byroyalty` `GATEWAY.pubs2..byroyalty`	byroyalty はデータベース所有者によって所有される。
`GATEWAY...byroyalty`	pubs2 がデフォルト・データベースの場合に使用する。
`declare @var int` `exec GATEWAY...byroyalty`	文がバッチ内の最初の文でない場合に使用する。

リモート・アクセスを行うための Adaptive Server の設定方法については、『システム管理ガイド 第 1 巻』の「第 15 章 リモート サーバの管理」を参照してください。リモート・サーバ名 (前述の例では GATEWAY) は、使用しているローカル Adaptive Server の *interfaces* ファイルにあるサーバ名と一致する必要があります。*interfaces* にあるサーバ名が大文字の場合は、RPC でも大文字を使用してサーバ名と一致させてください。

Adaptive Server の式

「式」は、演算子で区切られた 1 つ以上の定数、リテラル、関数、カラム識別子、および変数の組み合わせで、単独の値を返します。式には、「算術式」、「関係式」、「論理式 (またはブール式)」、および「文字列式」など、いくつかの種類があります。Transact-SQL 句には、式中にサブクエリを使用できるものがあります。case 式は、式中に使用できます。

式中の要素をグループ化するには、カッコを使用します。構文内で *expression* が変数として指定されている場合は、単純式が想定されます。1 つの論理式だけが受け入れ可能な場合は、logical_expression を使用します。

算術式と文字式

Adaptive Server には、複数の算術式および文字式があります。

演算子の優先度

算術演算子の優先度は次のとおりです。優先度の低い演算子から順に示します。

1　単項 (単独の引数) - + ~

2　* /%

3　2 項 (2 つの引数) + - & | ^

4　not

5　and

6　or

式中のすべての演算子が同じレベルの場合は、左から右の順で実行されます。実行の順序は、カッコを使用して変更します。最も内側にネストされた式が最初に実行されます。

算術演算子

Adaptive Server では、次の算術演算子を使用します。

表 1-5: 算術演算子

演算子	意味
+	加算
–	減算
*	乗算
/	除算
%	モジュロ (Transact-SQL 拡張機能)

加算、減算、除算、乗算は、真数値型、概数値型、および money 型のカラムで使用します。

money および numeric を除く真数カラムで使用できるモジュロ演算子は、2 つの数値の除算後の剰余を計算します。たとえば整数を使用する場合、21 を 11 で除算した商は 1 で剰余が 10 ですから、21 % 11 = 10 となります。numeric または decimal データ型では、整数でない結果が得られます。1.2 / 0.07 = 17 * 0.07 + 0.01 であるため、1.2 % 0.07 = 0.01 になります。float および real データ型の計算 1.2e0 % 0.07 = 0.010000 でも、同様の結果が返されます。

混合データ型 (たとえば float と int など) で算術演算を行うと、Adaptive Server は特定の規則に従って結果の型を決定します。「第 6 章 データ型の使用と作成」を参照してください。

ビット処理演算子

ビット処理演算子はデータ型 integer で使用するための Transact-SQL 拡張機能です。これらの演算子は整数のオペランドをそれぞれのバイナリ表現に変換して、カラムごとにオペランドを評価します。値 1 は true に対応します。値 0 は false に対応します。

表 1-6 と表 1-7 に、0 と 1 のオペランドの結果を示します。どちらかのオペランドが null の場合は、ビット処理演算子は null を返します。

表 1-6: ビット処理演算の真理値表

& (and)	1	0
1	1	0
0	0	0

| (or)	1	0
1	1	1
0	1	0

^ (exclusive or)	1	0
1	0	1
0	1	0

~ (not)	
1	FALSE
0	0

次の例では、2 つの tinyint 引数 A = 170
(バイナリ形式では10101010) および B = 75 (バイナリ形式では01001011) を使用しています。

表 1-7: ビット処理演算の例

演算	バイナリ形式	結果	説明
(A & B)	10101010 01001011 ---------- 00001010	10	A と B の両方が 1 のときは、結果カラムには 1 が表示される。それ以外の場合は、結果カラムには 0 が表示される。
(A | B)	10101010 01001011 ---------- 11101011	235	A と B のいずれか片方または両方が 1 のときは、結果カラムには 1 が表示される。それ以外の場合は、結果カラムには 0 が表示される。
(A ^ B)	10101010 01001011 ---------- 11100001	225	A と B 両方ではなく、いずれかが 1 の場合、結果カラムは 1。
(~A)	10101010 ---------- 01010101	85	1 がすべて 0 に変換され、すべての 0 は 1 に変更される。

文字列連結演算子

文字列演算子 + は、2 つ以上の文字式またはバイナリ式を連結できます。次に例を示します。

```
select Name = (au_lname + ", " + au_fname)
    from authors
```

これは、作家の名前をカラム見出し “Name” の下に姓、名前の順で表示します。姓の後にはカンマを置きます。たとえば “Bennett, Abraham” のようになります。

```
select "abc" + "" + "def"
```

これは文字列 “abc def” を返します。char、varchar、nchar、nvarchar、text 連結のすべて、および varchar 挿入文と代入文で、空文字列はシングル・スペースとして解釈されます。

文字以外の非バイナリ式を連結するときは、convert を使用します。

```
select "The date is " +
    convert(varchar(12), getdate())
```

比較演算子

Adaptive Server では、次の比較演算子を使用します。

表 1-8: 比較演算子

演算子	意味
=	等しい
>	より大きい
<	より小さい
>=	以上
<=	以下
<>	等しくない
!=	等しくない (Transact-SQL 拡張機能)
!>	より大きくない (Transact-SQL 拡張機能)
!<	より小さくない (Transact-SQL 拡張機能)

文字データの比較では、< はサーバのソート順の初めの方に近いことを意味し、>はソート順の終わりの方に近いことを意味します。大文字と小文字を区別しないソート順では、大文字と小文字は同等です。使用している Adaptive Server のソート順を表示するには、sp_helpsort を使用します。比較の場合、後続ブランクは無視されます。

日付の比較では、< は「より前」を意味し、> は「より後」を意味します。

比較演算子とともに使用するすべての文字と date や time のデータは、一重引用符か二重引用符で囲みます。

```
= "Bennet"
  "May 22 1947"
```

標準以外の演算子

次に示す演算子は Transact-SQL 拡張機能です。

- モジュロ演算子：(%)
- 否定の比較演算子：!>、!<、!=
- ビット処理演算子：~、^、|、&
- ジョイン演算子：*= および =*

文字式の比較

Adaptive Server は文字定数式を varchar として扱います。varchar 以外の変数やカラム・データと比較する場合、比較にはデータ型の優先度の規則が適用されます (つまり、優先度の低いデータ型が優先度の高いデータ型に変換されます)。暗黙的なデータ型の変換がサポートされていない場合は、convert 関数を使用する必要があります。サポートされる変換とサポートされない変換の詳細については、『リファレンス・マニュアル：ビルディング・ブロック』を参照してください。

char 式と varchar 式との比較は、データ型の優先度の規則に従います。「低い」データ型は「高い」データ型に変換されます。すべての varchar 式は、比較のために char に変換されます (つまり、後続ブランクが追加されます)。

空の文字列

空文字列 (“”) または (‘’) は、varchar データの insert 文や代入文では 1 つのブランクとして解釈されます。varchar、char、nchar、nvarchar データの連結では、空文字列はシングル・スペースとして解釈されます。たとえば、この文は文字列 “abc def” として格納されます。

```
"abc" + "" + "def"
```

空文字列は null として評価されることはありません。

引用符

char または varchar エントリ内にリテラル引用符を指定するには、2 つの方法があります。1 つは、同じ種類の引用符を追加する方法です。これは引用符の「エスケープ」と呼ばれます。たとえば、文字エントリを一重引用符で開始したが、エントリの一部に一重引用符を含みたい場合は、一重引用符を 2 つ使用します。

```
'I don''t understand.'
```

次に、両端の引用符の間に二重引用符と一重引用符がある例を示します。一重引用符はエスケープする必要はありませんが、二重引用符は次のようにエスケープする必要があります。

```
"He said, ""It's not really confusing."""
```

2 つ目は、もう一方の種類の引用符で、引用符を囲む方法です。つまり、二重引用符を含むエントリを一重引用符で (または一重引用符を含むエントリを二重引用符で) 囲みます。例を示します。

```
'George said, "There must be a better way."'
"Isn't there a better way?"
'George asked, "Isn"t there a better way?"'
```

画面の幅より長い文字列を入力するには、次の行に進む前に円記号 (¥) を入力します。

注意　quoted_identifier オプションが on に設定されている場合は、文字または日付データを二重引用符で囲まないでください。この場合は一重引用符を使用してください。そうしないと、Adaptive Server はデータを識別子として扱います。引用符で囲んだ識別子の詳細については、「区切り識別子」(12 ページ) を参照してください。

関係式と論理式

論理式または関係式は、TRUE (真)、FALSE (偽)、UNKNOWN (未知または認識できない) のいずれかを返します。一般的なパターンを示します。

expression *comparison_operator* [any | all] *expression*
expression [not] in *expression*
[not] exists *expression*
expression [not] between *expression* and *expression*
expression [not] like "*match_string*" [escape "*escape_character*"]
not expression like "*match_string*" [escape "*escape_character*"]
expression is [not] null
not *logical_expression*
logical_expression {and | or}*logical_expression*

any、*all*、*in*

any は <、>、または = のいずれかと、サブクエリとともに使用します。これは、サブクエリ内で検索された値のいずれかが、外側の文の where 句または having 句の値と一致すると、結果を返します。all は < か > のいずれかと、サブクエリとともに使用します。これは、サブクエリ内で検索されたすべての値が、外側の文の where 句または having 句の値より小さい (<) か、または大きい (>) と、結果を返します。「第 5 章 サブクエリ：他のクエリ内でのクエリの使用」を参照してください。

in は、2 番目の式が返した値のいずれかが、1 番目の式の値と一致する場合に、結果を返します。2 番目の式は、サブクエリ、またはカッコで囲んだ値のリストです。in は = any と等しくなります。

and と *or*

and は 2 つの式を接続し、両方が true の場合に結果を返します。or は 2 つ以上の条件を接続し、いずれかの条件が true であれば、結果を返します。

1 つの文中で複数の論理演算子が使用されるときは、or の前に and が評価されます。実行の順序は括弧を使用して変更できます。

表 1-9に、null 値を使用するものも含めて、論理演算の結果を示します。

表 1-9: 論理式の真理値表

and	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
NULL	UNKNOWN	FALSE	UNKNOWN
or	**TRUE**	**FALSE**	**NULL**
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
NULL	TRUE	UNKNOWN	UNKNOWN
not			
TRUE	FALSE		
FALSE	TRUE		
NULL	UNKNOWN		

UNKNOWN という結果は、1 つ以上の式が null と評価されて、演算の結果が TRUE であるか FALSE であるか判別できないことを示します。

Transact-SQL 拡張機能

Transact-SQL は SQL の能力を強化して、ユーザが希望のタスクを達成するのにプログラム言語に頼らなければならない場合を、最小限に抑えます。Transact-SQL は ISO 規格、および多くの商用版 SQL より優れた機能を持っています。

Transact-SQL 強化機能 (拡張機能) のほとんどは、ここにまとめられています。各コマンドの Transact-SQL 式は、『リファレンス・マニュアル：コマンド』に記載されています。

compute 句

Transact-SQL compute 句拡張機能は、ロー集合関数 sum、max、min、avg、count、count_big とともに使用され、合計値を計算します。compute 句を含むクエリの結果は、ディテール・ローおよびサマリ・ローの両方で表示されます。それらはレポート・ジェネレータを持つほとんどのデータベース管理システム (DBMS) で生成されるレポートと似ています。compute は計算値を新しいカラムとしてではなく、追加のローとして結果に表示します。compute 句については、「第 3 章 集合、グループ化、ソートの使用」を参照してください。

フロー制御言語

Transact-SQL は、任意の SQL 文またはバッチの一部として使用できるフロー制御言語を備えています。使用可能な構成体は、begin...end、break、continue、declare、goto label、if...else、print、raiserror、return、waitfor、while です。declare でローカル変数を定義し、値を割り当てることができます。多数の事前定義済みグローバル変数は、システムによって提供されます。

Transact-SQL は case 式もサポートしています。これにはキーワード case、when、then、coalesce、nullif が含まれます。case 式は標準の SQL の if 文に代わるものです。case 式は、値式が使用される箇所であればどこでも使用できます。

ストアド・プロシージャ

最も重要な Transact-SQL 拡張機能の 1 つは、ストアド・プロシージャを作成する機能です。「ストアド・プロシージャ」は、ある名前で格納されている SQL 文とオプションのフロー制御文の集合です。ストアド・プロシージャの作成者は、ストアド・プロシージャの実行時にパラメータが提供されるよう定義することもできます。

ストアド・プロシージャを独自に作成する機能によって、SQL データベース言語の能力、効果、柔軟性が大幅に強化されます。実行プランはストアド・プロシージャの実行後に保存されるので、その後は独立した文よりも高速に実行することができます。

Adaptive Server が提供するストアド・プロシージャは「システム・プロシージャ」と呼ばれ、Adaptive Server システム管理において使用します。「第 17 章 ストアド・プロシージャの使用」では、システム・プロシージャについて説明し、ストアド・プロシージャを作成する方法を説明しています。システム・プロシージャについては、『リファレンス・マニュアル：プロシージャ』で詳しく説明しています。

ユーザはリモート・サーバでストアド・プロシージャを実行できます。すべての Transact-SQL 拡張機能は、ストアド・プロシージャからの戻り値、ストアド・プロシージャからのユーザ定義リターン・ステータス、およびプロシージャからのパラメータをその呼び出し元に渡す機能をサポートしています。

拡張ストアド・プロシージャ

「拡張ストアド・プロシージャ」(ESP) では、ストアド・プロシージャと同じインタフェースを使用しますが、SQL 文とフロー制御文を含む代わりに、ダイナミック・リンク・ライブラリ (DLL) にコンパイルされた手続き型言語コードを実行します。

ESP 関数が作成される手続き型言語は、C 言語関数の呼び出しおよび C データ型の操作が可能な任意の言語です。

ESP によって、Adaptive Server は、データベース内で発生しているイベントに応じて、リレーショナル・データベース・マネジメント・システム (RDBMS) の外部のタスクを実行できます。たとえば、RDBMS 内で発生しているイベントに応じて、電子メールの通知やネットワーク全体に通知するために ESP を使用できます。

「システム拡張ストアド・プロシージャ」と呼ばれる、Adaptive Server が提供する ESP がいくつかあります。このうちの 1 つが xp_cmdshell で、これによってユーザは Adaptive Server 内からオペレーティング・システム・コマンドを実行できます。「第 18 章 拡張ストアド・プロシージャの使用」では、ESP について説明しています。『リファレンス・マニュアル：プロシージャ』の「第 3 章 システム拡張ストアド・プロシージャ」も参照してください。

ESP は、Adaptive Server と同じマシン上で実行される Open Server™ アプリケーションである XP Server™ によって実装されます。ストアド・プロシージャをリモートから実行することは「リモート・プロシージャ・コール」(RPC) と呼ばれます。Adaptive Server と XP Server は、RPC を介して通信します。XP Server は、Adaptive Server とともに自動的にインストールされます。

トリガ

「トリガ」とは、ある特定の変更が行われようとしたときに 1 つまたは複数のアクションを行うようシステムに命令するストアド・プロシージャです。トリガは、データへの不正な変更、認可されていない変更、または一貫性のない変更を防ぐことによって、データベースの整合性の維持に役立っています。

トリガは参照整合性も保護して、異なるテーブルでのデータ間の関係に関するルールを実行します。トリガは、ユーザが insert、delete、または update コマンドでデータを修正しようとすると起動します。

トリガは 16 レベルまでネストでき、ローカルまたはリモートのストアド・プロシージャを呼び出したり、別のトリガを呼び出したりすることができます。

「第 20 章 トリガ：参照整合性」を参照してください。

デフォルトとルール

Transact-SQL には、値を必要とするどのカラムにも値が確実に提供されるようにするためのエンティティの整合性を維持するためのキーワード、およびカラムの各値が、そのカラムに有効な値のセットに確実に属するようにするためのドメイン整合性を維持するためのキーワードが用意されています。デフォルトとルールは、データの入力や修正の間に使用される整合性制約を定義します。

デフォルトは特定のカラムやデータ型にリンクされた値で、データ入力中に値が提供されなかった場合に、システムによって挿入されます。ルールは特定のカラムやデータ型にリンクされたユーザ定義の整合性制約で、データ入力時に実行されます。ルールとデフォルトの詳細については、「第 14 章 データのデフォルトとルールの定義」を参照してください。

エラー処理と *set* オプション

Transact-SQL のエラー処理技法としては、たとえば、ストアド・プロシージャからのリターン・ステータスを取得する機能、ストアド・プロシージャからのカスタマイズされた戻り値を定義する機能、プロシージャからのパラメータを呼び出し元へ渡す機能、*@@error* などのグローバル変数からのレポートを取得する機能などがあります。raiserror および print 文は、フロー制御言語と組み合わせて、エラー・メッセージを Transact-SQL アプリケーションに送信できます。開発者は、異なる言語を使用するように print と raiserror をローカライズできます。

set オプションは結果の表示をカスタマイズして処理中の統計を表示し、Transact-SQL プログラムのデバッグをサポートする診断機能を提供します。showplan と char_convert を除くすべての set オプションは、ただちに有効になります。

『リファレンス・マニュアル：コマンド』を参照してください。

SQL のその他の Adaptive Server 拡張機能

Transact-SQL には、SQL に対する次の拡張機能があります。

- SQL 探索条件への拡張 (モジュロ演算子 (%)、否定の比較演算子 (!>、!<、!=)、ビット処理演算子 (–、^、|、&)、ジョイン演算子 (*=、=*)、ワイルドカード文字 ([ ] および -)、not 演算子 (^) など)。「第 2 章 クエリ：テーブルからのデータの選択」を参照してください。
- group by 句と order by 句に対する制限の緩和。「第 3 章 集合、グループ化、ソートの使用」を参照してください。
- サブクエリ。式を使用できる箇所であればほとんどの場合使用できます。「第 5 章 サブクエリ：他のクエリ内でのクエリの使用」を参照してください。
- テンポラリ・テーブルとその他のテンポラリ・データベース・オブジェクト。現在の作業セッションの間だけ存在します。「第 8 章 データベースおよびテーブルの作成」を参照してください。
- Adaptive Server 提供のデータ型上に構築されたユーザ定義データ型。「第 6 章 データ型の使用と作成」と「第 14 章 データのデフォルトとルールの定義」を参照してください。
- あるテーブルから同じテーブルへのデータの挿入 (insert)。「第 7 章 データの追加、変更、転送、削除」を参照してください。
- update コマンドを使用した、テーブルからのデータの抽出と、別のテーブルへの挿入。「第 7 章 データの追加、変更、転送、削除」を参照してください。
- delete 文中でジョインを使用した、他のテーブルのデータに基づくデータ削除。「第 7 章 データの追加、変更、転送、削除」を参照してください。

- truncate table コマンドを使用した、指定されたテーブルのすべてのローの削除と、そのローが使用していた領域の解放。「第 7 章 データの追加、変更、転送、削除」参照してください。
- identity カラム。テーブル内の各ローをユニークに識別するシステム生成値を提供します。「第 7 章 データの追加、変更、転送、削除」を参照してください。
- ビューを介した更新と選択。SQL の他の多くのバージョンとは異なり、Transact-SQL ではビューを介したデータ検索に制限はなく、ビューを介したデータの更新にもほとんど制限がありません。「第 12 章 ビュー：データへのアクセスの制限」を参照してください。
- 多数の組み込み関数。「第 16 章 クエリでの Transact-SQL 関数の使用」を参照してください。
- インデックスが決定するパフォーマンスを微調整し、重複するキーやローの処理を制御するための create index コマンドへのオプション。「第 13 章 テーブルのインデックスの作成」を参照してください。
- ユニーク・インデックスに重複するキーを入力しようとしたとき、またはテーブルに重複するローを入力しようとしたときの動作についてのユーザ制御。「第 13 章 テーブルのインデックスの作成」を参照してください。
- integer 型カラムおよび bit 型カラムで使用するためのビット処理演算子。「ビット処理演算子」(18 ページ)と「第 6 章 データ型の使用と作成」を参照してください。
- text データ型および image データ型のサポート。「第 6 章 データ型の使用と作成」を参照してください。
- Sybase データベースと Sybase 以外のデータベースの両方へのアクセス。コンポーネント統合サービスによって、ローカル・テーブルと同様のリモート・テーブルへのアクセス、ジョインの実行、テーブル間のデータの転送、参照整合性の維持、異機種データへの透過アクセス可能な PowerBuilder® などのアプリケーションの提供、ネイティブのリモート・サーバ機能の使用を行うことができます。詳細については、『コンポーネント統合サービス・ユーザーズ・ガイド』を参照してください。

Adaptive Server ログイン・アカウント

各 Adaptive Server ユーザには、システム・セキュリティ担当者によって確立されるログイン・アカウントが必要です。ログイン・アカウントには、次のものがあります。

- そのサーバ上でユニークなログイン名。

- パスワード。パスワードは定期的に変更することをおすすめします。『システム管理ガイド 第 1 巻』の「第 14 章 Adaptive Server のログイン、データベース・ユーザ、クライアント接続の管理」を参照してください。
- (オプション) デフォルト・データベース。デフォルト・データベースが定義されていると、各 Adaptive Server セッションは、use コマンドを発行しなくてもその定義されたデータベースで開始される。デフォルト・データベースが定義されていないと、各セッションは master データベースで開始される。
- (オプション) デフォルト言語。プロンプトとメッセージを表示する際の言語を指定する。デフォルト言語が定義されていないと、インストール時に設定された Adaptive Server のデフォルト言語が使用される。
- (オプション) フル・ネーム。ユーザのフルネーム。記録用と識別用として役立つ。

sp_displaylogin を使用して、自分の Adaptive Server ログイン・アカウントの情報を確認します。

グループを使用してデータベース内の複数のユーザに同時にパーミッションを付与したり取り消したりできます。たとえば、営業部門で働くすべての人があるテーブルにアクセスする必要がある場合、彼ら全員を “sales” というグループに入れることができます。データベース所有者は、各ユーザに個別にパーミッションを付与するのではなく、そのグループに特定のアクセス・パーミッションを付与できます。『システム管理ガイド 第 1 巻』の「第 14 章 Adaptive Server のログイン、データベース・ユーザ、クライアント接続の管理」を参照してください。

システム・セキュリティ担当者は、複数のユーザに同時にサーバ全体にわたるパーミッションを付与したり取り消したりするための便利な方法として、役割を使用できます。たとえば事務スタッフは、数あるデータベースの中のテーブルに対して挿入や選択を実行する必要があっても、更新を実行することは必要としない場合があります。システム・セキュリティ担当者は “clerical_user_role” という役割を定義して、事務スタッフの全員にその役割を付与できます。これで、データベース・オブジェクト所有者は “clerical_user_role” に必要な権限を付与できるようになります。『システム管理ガイド 第 1 巻』の「第 13 章 Adaptive Server のセキュリティ管理について」を参照してください。

リモート・サーバとそのサーバにある適切なデータベースへのアクセス権を付与されている場合には、リモート・プロシージャ・コールを使用してリモート Adaptive Server にあるストアド・プロシージャを実行できます。『システム管理ガイド 第 1 巻』の「第 15 章 リモート サーバの管理」を参照してください。

isql ユーティリティ

スタンドアロン・ユーティリティ・プログラムの isql を使用して、Transact-SQL 文をオペレーティング・システムから直接入力します。

まず、Adaptive Server のアカウント、つまりログイン情報を設定する必要があります。isql を使用するには、オペレーティング・システムのプロンプトで次のようなコマンドを入力します。

```
isql -Uuser_name -Ppassword -Sserver_name
```

ログインすると、次のような表示になります。

```
1>
```

注意 コマンド・ラインで -P オプションを使用して isql にアクセスしないでください。むしろ、別のユーザにパスワードを見られないようにするため、isql パスワード・プロンプトを待ちます。

次のように入力して、isql からログアウトします。

```
quit
```

または

```
exit
```

『ユーティリティ・ガイド』を参照してください。

コンポーネント統合サービスを使用して Sybase 以外のデータベースに接続するには、connect to コマンドを使用します。コンポーネント統合サービス・ユーザーズ・ガイド を参照してください。『リファレンス・マニュアル：コマンド』の「connect to...disconnect」も参照してください。

デフォルト・データベース

Adaptive Server アカウントが作成されたときに、ユーザがログインすると接続されるデフォルト・データベースが割り当てられる場合があります。たとえば、サンプル・データベースの pubs2 がデフォルト・データベースの場合があります。デフォルト・データベースが割り当てられなかった場合は、「**master** データベース」に接続します。

デフォルト・データベースは、使用するパーミッションを持っている任意のデータベース、または guest を許可する任意のデータベースに変更できます。Adaptive Server ログインを持つユーザであれば誰でも guest になることができます。デフォルト・データベースを変更するには、sp_modifylogin を使用します。このプロシージャについては『リファレンス・マニュアル：プロシージャ』で説明しています。

このマニュアル内のほとんどの例で使用される pubs2 データベースに変更するには、次のように入力します。

```
1> use pubs2
2> go
```

“go” は 1 行に単独で入力し、その前にブランクやタブを付けません。これはコマンド・ターミネータであり、ユーザが入力を終了し、コマンドを実行する準備ができていることを Adaptive Server に通知するものです。

一般的に、このマニュアルで示す Transact-SQL 文の例には、isql ユーティリティが使用するプロンプトも go ターミネータも含みません。

isql でのネットワークベース・セキュリティ・サービス

isql の -V オプションを指定して、統一化ログインなどのネットワークベース・セキュリティ・サービスを使用できます。統一化ログインを使用すると、ユーザはサード・パーティのプロバイダによって提供されるセキュリティ・メカニズムで認証され、ログイン名やパスワードを指定せずに Adaptive Server にログインできます。

ネットワークベース・セキュリティを使用する場合に指定できるオプションの詳細については、『ユーティリティ・ガイド』および『システム管理ガイド 第 1 巻』の「第 16 章 外部認証」を参照してください。

SQL テキストの表示

set show_sqltext を使用すると、アドホック・クエリ、ストアド・プロシージャ、カーソル、動的 prepared 文の SQL テキストを出力できます。set showplan on などのコマンドで行うように、クエリを実行して SQL セッションの診断情報を収集する前に、set show_sqltext を有効にする必要はありません。その代わりに、各コマンドの実行中にこのコマンドを有効にして、どのクエリが適切に実行されていないかを判断できます。

set show_sqltext を有効にする前に dbcc traceon を有効にして、コマンド結果を標準出力 (`stdout`) に送ります。

```
dbcc traceon(3604)
```

set show_sqltext の構文は、次のとおりです。

```
set show_sqltext {on | off}
```

たとえば、次の例は show_sqltext を有効にします。

```
set show_sqltext on
```

set show_sqltext が有効になると、入力した各コマンドとシステム・プロシージャに対するすべての SQL テキストが `stdout` に出力されます。実行するコマンドまたはシステム・プロシージャに応じて、この出力は長くなる場合があります。

show_sqltext を無効にするには、次のように入力します。

```
set show_sqltext off
```

show_sqltext の制限

- show_sqltext を実行するには、sa_role または sso_role が必要です。
- show_sqltext を使用してトリガの SQL テキストを出力することはできません。
- show_sqltext を使用して、バインド変数または表示名を示すことはできません。

第 2 章

クエリ：テーブルからのデータの選択

select コマンドでは、クエリと呼ばれるプロシージャを使用して、データベース・テーブルのローとカラムに保存されているデータを取り出すことができます。クエリは、select 句、from 句、where 句という 3 つの主要部分を持ちます。

この章では基本的な単一テーブルの select 文を中心に説明します。項の多くには、クエリの作成に使用できる文のサンプルが用意されています。ジョイン、サブクエリ、集約など、他の Transact-SQL 機能の組み込みについては、このマニュアルの後半でより複雑なクエリの例を挙げています。

クエリ

SQL クエリはデータベースからデータを要求します。このプロセスは「データ検索」とも呼ばれ、select 文を使用して表されます。これは、1 つ以上のテーブルのローのサブセットを取り出す「選択」に使用したり、1 つ以上のテーブルのカラムのサブセットを取り出す「射影」に使用したりできます。

select 文の簡単な例を次に示します。

```
select select_list
from table_list
where search_conditions
```

select 句は、検索するカラムを指定します。from 句は、検索するテーブルを指定します。where 句では、テーブルのどのローを参照するかを指定します。たとえば、次の select 文は、オークランドに居住する作家の氏名を pubs2 データベースの authors テーブルから検索します。

```
select au_fname, au_lname
from authors
where city = "Oakland"
```

このクエリの結果は、次のようにカラム・フォーマットで表示されます。

```
au_fname        au_lname
--------------  -----------
Marjorie        Green
Dick            Straight
Dirk            Stringer
Stearns         MacFeather
Livia           Karsen

(5 rows affected)
```

select 構文

select 構文は、前述の例よりも単純にすることも、複雑にすることもできます。単純な select 文は select 句しか含みません。from 句はほとんど常に含まれますが、これはテーブルからデータを検索する select 文でのみ必要です。その他のすべての句 (where 句も含む) はオプションです。

select 文の完全な構文については、『リファレンス・マニュアル：コマンド』を参照してください。

TOP *unsigned_integer* を使用すると、結果セット内のローの数を制限できます。表示されるローの数を指定します。TOP は、同じ目的で、delete コマンドおよび update コマンドでも使用されます。『リファレンス・マニュアル：コマンド』を参照してください。

select 文内の句は、ここに示した順で使用します。たとえば文に group by 句と order by 句が含まれる場合、group by 句は order by 句の前になければなりません。

参照されているオブジェクトがあいまいである場合は、データベース・オブジェクトの名前を修飾します。“name” というカラムが複数のテーブル内にいくつか存在する場合は、データベース名や所有者名、またはテーブル名などで “name” を修飾する必要があります。次に例を示します。

```
select au_lname from pubs2.dbo.authors
```

この章の例は単一テーブルでのクエリを扱っているので、構文モデルおよび例のカラム名は、通常はそのカラムが属するテーブル、所有者、またはデータベースなどの名前で修飾されていません。見やすくするためにこのような要素を省いていますが、修飾子を含めても間違いではありません。この章の以降の項で、select 文の構文をより詳細に分析します。

この章では、select コマンドの構文に含まれる句およびキーワードの一部についてのみ説明します。次に示す句は、他の章で説明します。

- group by、having、order by、compute は、「第 3 章 集合、グループ化、ソートの使用」で説明します。
- into は、「第 8 章 データベースおよびテーブルの作成」で説明します。
- at isolation は、「第 23 章 トランザクション：データの一貫性およびリカバリ」で説明します。

holdlock、noholdlock、shared キーワード (Adaptive Server のロックを処理する) と index 句は、『パフォーマンス＆チューニング・シリーズ：ロックと同時実行制御』の「第 4 章 ロック・コマンドの使用」で説明します。for read only 句および for update 句の詳細については、『リファレンス・マニュアル：コマンド』の「declare cursor コマンド」の項を参照してください。

注意　for browse 句は DB-Library アプリケーション内でだけ使用されます。詳細については、『Open Client DB-Library/C リファレンス・マニュアル』を参照してください。「カーソルの代わりとしてのブラウズ・モードの使用」(604 ページ) も参照してください。

---

select 文の識別子の確認

ストアド・プロシージャまたはトリガのソース・テキストが syscomments システム・テーブルに格納される場合、select * を使用するクエリは、select * で参照されるカラム・リストを拡張する syscomments にも格納されます。

たとえば、カラム col1 と col2 を含むテーブルの select * は、次のように格納されます。

```
select <table>.col1, <table>.col2 from <table>
```

カラム・リストは、識別子 (テーブル名、カラム名など) が識別子のルールに準じていることを確認します。

たとえば、テーブルにカラム col1 と 2col が含まれる場合、2 番目のカラム名は数値で始まります。この 2 番目のカラム名は create table 文でカッコに入れる必要があります。

このテーブルから、ストアド・プロシージャまたはトリガで select * を実行すると、syscomments 内のテキストは次のようになります。

```
select <table>.col1, <table>[2col] from <table>
```

select * を拡張したテキストで使用されるすべての識別子で、識別子が識別子のルールに従わない場合は角カッコが追加されます。

select 句によるカラムの選択

select 句の項目によって、select リストが構成されます。select リストにカラム名、カラムのグループ、またはワイルドカード文字 (*) が含まれている場合、データは、テーブルに格納されている順序 (create table 時の順序) で取り出されます。

select * によるすべてのカラムの選択

アスタリスク (*) を使用すると、from 句によって指定されるすべてのテーブル内のすべてのカラム名が選択されます。テーブル内のすべてのカラムを参照するときは、これを使用して入力の手間とエラーを省きます。* を使用すると、create table での記述順でデータが取り出されます。

テーブル内のすべてのカラムを選択する構文を次に示します。

```
select *
from table_list
```

次の文は、publishers テーブル内のすべてのカラムを取り出し、それを create table の記述順で表示します。この文には、where 句が含まれていないため、すべてのローが取り出されます。

```
select *
from publishers
```

結果は次のようになります。

```
pub_id  pub_name              city          state
------  --------------        ---------     -----
0736    New Age Books         Boston        WA
0877    Binnet & Hardley      Washington    DC
1389    Algodata Infosystems  Berkeley      CA

(3 rows affected)
```

select キーワードの後に、テーブルのすべてのカラム名を順番に並べた場合も、まったく同じ結果が得られます。

```
select pub_id, pub_name, city, state
from publishers
```

クエリ内で “*” を複数回使用することもできます。

```
select *, *
from publishers
```

このクエリは、各カラム名とカラム・データの各部分を 2 回表示します。カラム名と同様、テーブル名もアスタリスクで修飾できます。次に例を示します。

```
select publishers.*
from publishers
```

ただし、select * は現在テーブルにあるすべてのカラムを検索するので、カラムの追加、削除、名前の変更など、テーブルの構造に変更があると、select * の結果は自動的に修正されます。カラムを個別にリストした方が、結果をより精密に制御できます。

特定のカラムの選択

テーブルの特定のカラムだけを選択するには、カラム名をカンマで区切って次の構文を使用します。

```
select column_name[, column_name]...
    from table_name
```

次に例を示します。

```
select au_lname, au_fname
from authors
```

カラム順の並べ替え

select 句でカラム名をリストする順序によって、カラムを表示する順序が決定されます。次の例ではカラム順序の指定方法を示しています。publishers テーブルの 3 つのローすべてから取り出した出版社の名前と ID 番号を表示します。最初の例は pub_id を先に、次に pub_name を出力します。2 番目の例は逆の順になります。情報は同じですが、編成が変わります。

```
select pub_id, pub_name
from publishers

pub_id   pub_name
-----    ---------------
0736     New Age Books
0877     Binnet & Hardley
1389     Algodata Infosystems

(3 rows affected)

select pub_name, pub_id
from publishers

pub_name                       pub_id
---------------------          ------
New Age Books                  0736
Binnet & Hardley               0877
Algodata Infosystems           1389

(3 rows affected)
```

クエリ結果でのカラム名の変更

クエリ結果では、各カラムのデフォルトの見出しが、作成されたときに指定される名前になります。select リストでカラム名を使用する代わりに次のいずれかを使用することによって、カラム見出しの名前を見やすい表示にできます。

column_heading = *column_name*

column_name column_heading

column_name as *column_heading*

これによってカラムに代替の名前が提供されます。たとえば、前述のクエリで pub_name を “Publisher” に変更するには、次の文のいずれかを入力します。

```
select Publisher = pub_name, pub_id from publishers

select pub_name Publisher, pub_id from publishers

select pub_name as Publisher, pub_id from publishers
```

これらの文のいずれの結果も次のようになります。

```
Publisher                 pub_id
----------------------    ------
New Age Books             0736
Binnet & Hardley          0877
Algodata Infosystems      1389

(3 rows affected)
```

式の使用

select 文には、1 つまたは複数の「式」を含めることができます。これにより、取り出したデータを操作できます。

```
select expression [, expression]...
   from table_list
```

式は、定数、カラム名、関数、サブクエリ、case 式を算術演算子、ビット処理演算子、カッコで連結した任意の組み合わせです。

リスト内のいずれかのテーブルまたはカラムが有効な識別子の規則に従っていない場合は、quoted_identifier オプションを on に設定して、識別子を二重引用符で囲んでください。

カラム見出しの引用符付き文字列

見出し全体を引用符で囲めば、カラム見出しには、任意の文字を含めることができます。これにはブランクも含まれます。quoted_identifier オプションを on に設定する必要はありません。引用符で囲まれていない場合、カラム見出しは識別子の規則に従う必要があります。次の 2 つのクエリは、同じ結果を生成します。

```
select "Publisher's Name" = pub_name from publishers
```

```
select pub_name "Publisher's Name" from publishers

Publisher's Name
----------------
New Age Books
Binnet & Hardley
Algodata Infosystems

(3 rows affected)
```

また、引用符付きカラムの見出しには、Transact-SQL 予約語も使用できます。たとえば、カラム見出しに予約語 sum を使用した次のクエリは、有効です。

```
select "sum" = sum(total_sales) from titles
```

引用符付きカラムの見出しの長さは、255 バイトを超えることはできません。

注意　create table、alter table、select into、または create view 文でカラム名を引用符で囲む前に、quoted_identifier を on に設定してください。

クエリ結果の文字列

これまでの例の select は、データベース内のデータを示す結果を生成します。また、結果に文字列を含めることができるようにクエリを記述することもできます。含める文字列を一重または二重の引用符で囲み、カンマを使用して select リスト内の他の要素から区切ります。文字列にアポストロフィがある場合は二重引用符を使用します。そうしないと、アポストロフィが一重引用符として解釈されてしまいます。

次に文字列を使用するクエリを示します。

```
select "The publisher's name is", Publisher = pub_name
from publishers

                                    Publisher
------------------------            --------------------
The publisher's name is             New Age Books
The publisher's name is             Binnet & Hardley
The publisher's name is             Algodata Infosystems

(3 rows affected)
```

select リスト内の計算値

日付関数を使用して date/time カラムに対して実行できる算術演算があります。詳細については、「第 16 章 クエリでの Transact-SQL 関数の使用」を参照してください。これらの演算子はすべて、カラム名と数値定数を使用して任意に組み合わせて、select リスト内で使用できます。たとえば、titles テーブルのすべての本について、100 パーセントの売り上げ増加を計算するとどのようになるかを調べるには、次を入力します。

```
select title_id, total_sales, total_sales * 2
from titles
```

結果は次のようになります。

```
title_id     total_sales
--------     -----------       ---------
BU1032              4095            8190
BU1111              3876            7752
BU2075             18722           37444
BU7832              4095            8190
MC2222              2032            4064
MC3021             22246           44492
MC3026              NULL            NULL
PC1035              8780           17560
PC8888              4095            8190
PC9999              NULL            NULL
PS1372               375             750
PS2091              2045            4090
PS2106               111             222
PS3333              4072            8144
PS7777              3336            6672
TC3218               375             750
TC4203             15096           30192
TC7777              4095            8190

(18 rows affected)
```

total_sales カラムと計算カラムの null 値に注意してください。null 値には、明示的に割り当てられた値がありません。null 値に算術演算を実行すると、結果は null になります。次のように入力して、計算カラムに、たとえば “proj_sales” などのカラム見出しを指定します。

```
select title_id, total_sales,
     proj_sales = total_sales * 2
from titles
```

```
title_id     total_sales     proj_sales
---------    -----------     -----------
BU1032              4095            8190
 ....
```

“Current sales =” や “Projected sales are” などの文字列を select 文に追加してみてください。計算カラムが生成されるカラムは、select リストになくてもかまいません。たとえば total_sales カラムは、その値を total_sales * 2 カラムの値と比較するためにしかこれらのサンプル・クエリ内に示されていません。計算値だけを表示するには、次を入力します。

```
select title_id, total_sales * 2
from titles
```

算術演算子は、定数値が使用されていない場合、指定されたカラムのデータ値を直接演算します。次に例を示します。

```
select title_id, total_sales * price
from titles

title_id
--------     ----------
BU1032        81,859.05
BU1111        46,318.20
BU2075        55,978.78
BU7832        81,859.05
MC2222        40,619.68
MC3021        66,515.54
MC3026             NULL
PC1035       201,501.00
PC8888        81,900.00
PC9999             NULL
PS1372         8,096.25
PS2091        22,392.75
PS2106           777.00
PS3333        81,399.28
PS7777        26,654.64
TC3218         7,856.25
TC4203       180,397.20
TC7777        61,384.05

(18 rows affected)
```

計算カラムも複数のベース・テーブルから抽出できます。複数のテーブルのクエリについては、ジョインとサブクエリの章で説明します。

ジョインの例として、このクエリでは、書店が売り上げた心理学の本の部数 (salesdetail テーブルの qty カラム) とその本の価格 (titles テーブルの price カラム) の積を計算します。

```
select salesdetail.title_id, stor_id, qty * price
from titles, salesdetail
where titles.title_id = salesdetail.title_id
and titles.title_id = "PS2106"

title_id         stor_id
---------------- -----------  ------
PS2106           8042         210.00
PS2106           8042         350.00
PS2106           8042         217.00

(3 rows affected)
```

算術演算子の優先度

1 つの式に複数の算術演算子がある場合、乗算、除算、モジュロが先に計算され、続いて減算と加算が計算されます。式中のすべての算術演算子の優先度が同じレベルの場合は、左から右の順で実行されます。カッコで囲まれた式は、他のどの演算よりも優先されます。

たとえば次の select 文は、ある本の総売り上げ数にその価格を乗算して合計額を計算し、作家の前払い金の半分をそこから減算します。

```
select title_id, total_sales * price - advance / 2
from titles
```

演算子が乗法なので、total_sales と price の積が先に計算されます。次に前払い金が 2 で除算され、その結果が total_sales * price から減算されます。

誤解を招かないようにするには、カッコを使用します。次のクエリは前述のクエリと同じ意味で同じ結果を生成しますが、こちらの方がわかりやすくなっています。

```
select title_id,(total_sales * price) - (advance /2)
from titles

title_id
--------     ----------
BU1032        79,359.05
BU1111        43,818.20
BU2075        50,916.28
BU7832        79,359.05
MC2222        40,619.68
MC3021        59,015.54
MC3026             NULL
PC1035       198,001.00
PC8888        77,900.00
PC9999             NULL
PS1372         4,596.25
PS2091        21,255.25
PS2106        -2,223.00
PS3333        80,399.28
PS7777        24,654.64
TC3218         4,356.25
TC4203       178,397.20
TC7777        57,384.05

(18 rows affected)
```

実行の順序を変更するには、カッコを使用します。カッコ内の計算が最初に処理されます。カッコがネストされている場合は、最も内側にネストされた計算が優先されます。たとえば、カッコを使用して除算より先に減算を行うようにすると、前の例の結果と意味が変更されます。

```
select title_id, (total_sales * price - advance) /2
from titles
title_id
--------   -----------------------
BU1032                  38,429.53
BU1111                  20,659.10
BU2075                  22,926.89
BU7832                  38,429.53
MC2222                  20,309.84
MC3021                  25,757.77
```

```
MC3026                       NULL
PC1035                    97,250.50
PC8888                    36,950.00
PC9999                       NULL
PS1372                     548.13
PS2091                    10,058.88
PS2106                    -2,611.50
PS3333                    39,699.64
PS7777                    11,327.32
TC3218                       428.13
TC4203                    88,198.60
TC7777                    26,692.03

(18 rows affected)
```

text、*unitext*、*image* 値の選択

text、unitext、image 値は、非常に大きい場合があります。select リストに text、unitext、image 値が含まれる場合、返されるデータの長さの制限は、*@@textsize* グローバル変数の設定に依存します。*@@textsize* のデフォルト設定は、Adaptive Server へのアクセスに使用するソフトウェアによって異なります。isql の場合、デフォルト値は 32K です。値を変更するには、set コマンドを使用します。

```
set textsize 2147483648
```

この設定の *@@textsize* を使用すると、text カラムを含む select 文は、データの始めの 2 ギガバイトだけを表示します。

注意　image データを選択している場合、返される値には文字 “0x” が含まれます。これはデータが 16 進であることを示します。これらの 2 つの文字は *@@textsize* の一部としてカウントされます。

@@textsize を Adaptive Server のデフォルト値にリセットするには、次を使用します。

```
set textsize 0
```

返されたデータの実際の長さが textsize よりも短い場合、データ文字列全体が表示されます。「第 6 章 データ型の使用と作成」を参照してください。

readtext の使用

カラムに含まれるデータの特定の部分だけを取り出す場合は、readtext コマンドで text、unitext、および image の値を取り出すことができます。readtext コマンドには、テーブルおよびカラムの名前、テキスト・ポインタ、カラム内の開始オフセット、検索する文字数またはバイト数の情報が必要です。この例では、blurbs テーブルの copy カラム内で 6 文字を検索します。

```
declare @val binary(16)
select @val = textptr(copy) from blurbs
where au_id = "648-92-1872"
readtext blurbs.copy @val 2 6 using chars
```

この例では、*@val* ローカル変数が宣言された後、readtext は copy カラムの文字 3 ～ 8 (オフセットが 2 であるため) を表示します。

Adaptive Server では、サイズが大きくなる可能性のある text、unitext、image のデータをテーブルに格納する代わりに、特別な構造体に格納します。データが実際に格納されるページを指すテキスト・ポインタ (textptr) が割り当てられます。readtext を使用してデータを取り出すときは、実際には、textptr を取り出しています。これは、16 バイトの varbinary 文字列です。これを避けるには、上の例のように、textptr を保持するローカル変数を宣言してから、その変数を readtext とともに使用します。

readtext コマンドの詳細については、「text 関数および image 関数」(487 ページ) を参照してください。

select リストの概要

select リストには *(create table での記述順のすべてのカラム)、任意の順序でのカラム名のリスト、文字列、カラム見出し、算術演算子を含む式を含めることができます。

```
select titles.*
from titles


select Name = au_fname, Surname = au_lname
from authors


select Sales = total_sales * price,
ToAuthor = advance,
ToPublisher = (total_sales * price) - advance
from titles


select "Social security #", au_id
from authors


select this_year = advance, next_year = advance
   + advance/10, third_year = advance/2,
    "for book title #", title_id
from titles


select "Total income is",
```

```
Revenue = price * total_sales,
"for", Book# = title_id
from titles
```

集合関数を含めることもできます。これについては、「第 3 章 集合、グループ化、ソートの使用」で説明しています。

select for update の使用

Adaptive Server バージョン 15.7 以降では、同じトランザクション内の後続の更新、および更新可能なカーソルのためにデータローロック・テーブルの排他ロックを行うための select for update がサポートされています。これにより、同時に実行される他のタスクがこれらのローを更新したり、後続の更新をブロックすることを防止できます。select for update は独立性レベル 1、2、3 でサポートされています。

15.7 より前のバージョンでは、for update 句を使用する select 文は declare cursor 文内でのみ発行することができました。

Adaptive Server バージョン 15.7 以降では、select for update を、カーソル・コンテキストの外部の言語文として発行できます。言語文とカーソルのいずれの場合でも、begin transaction コマンドまたは連鎖モード内で select for update を実行する必要があります。

select for update をカーソル・コンテキストで実行する場合、カーソル open と fetch 文はトランザクションのコンテキスト内でなければなりません。そうでない場合は、Adaptive Server が 15.7 より前の機能に戻ります。

構文

```
select <col-list> from … where …
    [for update[ of col-list ]
```

注意　バージョン 15.7 の機能と排他ロックを使用するには、select for update 設定パラメータを 1 に設定し、for update 句を含める必要があります。この設定パラメータを設定しない場合、Adaptive Server reverts が 15.7 より前の機能に戻ります。

カーソルおよび DML での *select for update* の使用

select for update の機能は、設定パラメータ select for update の値に基づきます。

- 0 － 15.7 より前のバージョンの Adaptive Server の機能が適用され、select for update はカーソルでのみ使用できます。
- 1 － Adaptive Server バージョン 15.7 の機能が適用されます。select for update をカーソル・コンテキストの外部の言語レベルで使用できます。

15.7 よりも前のバージョンの場合：

- select for update はカーソルでのみサポートされます。
- ローを修飾し、ライタ (他のリーダを除く) をブロックするために更新ロックが必要になります。
- order by 句は、カーソルを自動的に読み取り専用にするため、更新可能カーソルに使用できません。

バージョン 15.7 以降の場合：

- select for update は言語文とカーソルでサポートされています。
- 次の場合、select for update のローを修飾して、他のリーダとライタをブロックするために、排他ロックが必要になります。
  - データローロック・テーブルを使用している
  - トランザクションのコンテキストや連鎖モードでコマンドを使用している
- select for update には、言語文とカーソルのいずれの場合も、order by 句を使用することができます。for update 句に order by 句を使用することにより、カーソルを更新可能にすることができます。

select for update の構文と使用方法、カーソルのスコープと使用方法については、『リファレンス・マニュアル：コマンド』を参照してください。

同時実行性に関する問題

セッションのオープン・トランザクションで、独立性レベル 1、2、または 3 で select for update を実行している場合、2 番目のセッションにより発行されるトランザクションのタイプとその独立性レベルによっては、同じテーブルにデータ操作言語 (DML) 文を発行する 2 番目の同時実行セッションがブロックされることがあります。

表 2-1: 2 番目の同時実行セッションのトランザクションの状態

	独立性レベル			
トランザクション	0	1	2	3
select の条件を満たすロー	ブロックなし	ブロックなし[1]	ブロック	ブロック
select の条件を満たさないロー	ブロックなし	ブロックなし	ブロックなし[2]	ブロックなし[2]

[1] Adaptive Server は、select リストに text、image、unitext などのラージ・オブジェクト (LOB) が含まれない限り、独立性レベル 1 の select コマンドをブロックしません。

[2] 最初のセッションで独立性レベル 3 の select for update が発行されている場合、Adaptive Server では、条件を満たすローよりも多くのローを排他ロックして「幻ロー」を防止します。この場合、Adaptive Server は、これらの条件を満たさない追加ローに対する 2 番目のセッションをブロックします。

[3] 2 番目のセッションで独立性レベル 0 の DML を発行した場合でも Adaptive Server により独立性レベル 2 で実行されます。

[4] Adaptive Server では、独立性レベル 0 の select for update をサポートしていません。

	独立性レベル			
トランザクション	**0**	**1**	**2**	**3**
update の条件を満たすロー	ブロック[3]	ブロック	ブロック	ブロック
update の条件を満たさないロー	ブロックなし[2,3]	ブロックなし[2]	ブロックなし[2]	ブロックなし[2]
select for update の条件を満たすロー	N/A[4]	ブロック	ブロック	ブロック
select for update の条件を満たさないロー	N/A[4]	ブロックなし[2]	ブロックなし[2]	ブロックなし[2]
delete の条件を満たすロー	ブロック[3]	ブロック	ブロック	ブロック
delete の条件を満たさないロー	ブロックなし[2,3]	ブロックなし[2]	ブロックなし[2]	ブロックなし[2]
insert	ブロックなし[2,3]	ブロックなし[2]	ブロックなし[2]	ブロックなし[2]

[1] Adaptive Server は、select リストに text、image、unitext などのラージ・オブジェクト (LOB) が含まれない限り、独立性レベル 1 の select コマンドをブロックしません。

[2] 最初のセッションで独立性レベル 3 の select for update が発行されている場合、Adaptive Server では、条件を満たすローよりも多くのローを排他ロックして「幻ロー」を防止します。この場合、Adaptive Server は、これらの条件を満たさない追加ローに対する 2 番目のセッションをブロックします。

[3] 2 番目のセッションで独立性レベル 0 の DML を発行した場合でも Adaptive Server により独立性レベル 2 で実行されます。

[4] Adaptive Server では、独立性レベル 0 の select for update をサポートしていません。

distinct による重複するクエリ結果の消去

オプションの distinct キーワードは、select 文のデフォルトの結果から重複するローを消去します。

SQL の他の実装との互換性のために、Adaptive Server 構文では、all を使用してすべてのローを明示的に要求することができます。select 文のデフォルトは all です。distinct を指定しない場合は、デフォルトでは、重複したローも含めすべてのローが取り出されます。

たとえば、distinct を使用しないで titleauthor テーブルにある作家の ID コードをすべて検索した場合の結果は以下のようになります。

```
select au_id
from titleauthor

au_id
-----------
172-32-1176
213-46-8915
213-46-8915
238-95-7766
267-41-2394
267-41-2394
274-80-9391
409-56-7008
427-17-2319
472-27-2349
```

```
486-29-1786
486-29-1786
648-92-1872
672-71-3249
712-45-1867
722-51-5454
724-80-9391
724-80-9391
756-30-7391
807-91-6654
846-92-7186
899-46-2035
899-46-2035
998-72-3567
998-72-3567

(25 rows affected)
```

重複するリストがいくつかあります。distinct を使用して重複を消去します。

```
select distinct au_id
from titleauthor

au_id
-----------
172-32-1176
213-46-8915
238-95-7766
267-41-2394
274-80-9391
409-56-7008
427-17-2319
472-27-2349
486-29-1786
648-92-1872
672-71-3249
712-45-1867
722-51-5454
724-80-9391
756-30-7391
807-91-6654
846-92-7186
899-46-2035
998-72-3567

(19 rows affected)
```

null 値が複数ある場合、distinct では重複として扱われます。つまり、select 文に distinct が含まれている場合は、null 値がいくつ検出されたかに関係なく、null は 1 つだけ返されます。

order by 句とともに使用すると、distinct は複数の値を返すことができます。「order by および group by の select distinct での使用」(93 ページ) を参照してください。

from 句によるテーブルの指定

from 句は、テーブルまたはビューに含まれるデータを検索するすべての select 文に必要です。from 句を使用して、select リストと where 句に含まれるカラムを持つテーブルやビューをすべてリストします。from 句が複数のテーブルやビューを指定する場合は、カンマで区切ります。

クエリは最大で 50 個のテーブルと 46 個のワーク・テーブル (集合関数によって作成されたテーブルなど) を参照できます。50 個のテーブル制限には次のものが含まれます。

- from 句にリストされるテーブル (またはテーブルのビュー)
- 同じテーブルに対する複数の参照 (セルフジョイン) の各インスタンス
- サブクエリで参照されるテーブル
- into で作成されるテーブル
- from 句にリストされるビューによって参照されるベース・テーブル

『リファレンス・マニュアル：コマンド』を参照してください。

テーブル名は 1～255 バイトの長さで指定することができます。最初の文字には、英字、@、#、_ を使用することができます。以降は、数字、英字、@、#、$、_、¥、または £ を使用することができます。テンポラリ・テーブル名は、tempdb の外部で作成された場合は “#” (シャープ記号) で、それ以外の場合は “tempdb..” で開始する必要があります。テンポラリ・テーブルの名前は 238 バイトを超えないようにしてください。Adaptive Server が、名前をユニークにするために名前に 17 バイトの内部数値サフィックスを付加するからです。「第 8 章 データベースおよびテーブルの作成」を参照してください。

from 句内では、次のような、テーブルおよびビューの完全な命名構文を常に使用することができます。

```
database.owner.table_name
    database.owner.view_name
```

ただし、これは、名前に混乱が発生しそうな場合にだけ必要です。

テーブル名に相関名を指定して、入力の手間を省くことができます。次のようにテーブル名の後に相関名を指定して、from 句で相関名を割り当てることができます。

```
select p.pub_id, p.pub_name
from publishers p
```

そのテーブルへの他のすべての参照 (たとえば where 句内での参照) も、この相関名を使用する必要があります。相関名は数字では開始できません。

where 句によるローの選択

select 文内の where 句は、検索するローの条件を指定します。一般的なフォーマットを示します。

```
select select_list
    from table_list
    where search_conditions
```

where 句の検索条件または修飾には、次のものがあります。

- 比較演算子 (=、<、> など)

  ```
  where advance * 2 > total_sales * price
  ```

- 範囲 (between および not between)

  ```
  where total_sales between 4095 and 12000
  ```

- リスト (in、not in)

  ```
  where state in ("CA", "IN", "MD")
  ```

- 文字の一致 (like および not like)

  ```
  where phone not like "415%"
  ```

- 不定の値 (is null および is not null)

  ```
  where advance is null
  ```

- 検索条件の組み合わせ (and、or)

  ```
  where advance < 5000 or total_sales between 2000
    and 2500
  ```

where キーワードには、次のものも導入できます。

- ジョイン条件 (「第 4 章 ジョイン：複数テーブルからのデータの検索」を参照)
- サブクエリ (「第 5 章 サブクエリ：他のクエリ内でのクエリの使用」を参照)

注意 text カラムに使用できる唯一の where 条件は、like (または not like) です。

Adaptive Server では、必ずしも左から右に述語を評価して実行するわけではありません。その代わり、あらゆる順序で述語を評価して実行する場合があります。たとえば、次のクエリがあるとします。

```
where x != 0 and y = 10 or z = 100
```

Adaptive Server は `x != 0` を最初に評価して実行しない場合があります。

探索条件の詳細については、『リファレンス・マニュアル：コマンド』を参照してください。

where 句の比較演算子

比較を行う場合には、後続ブランクは無視されます。たとえば、“Dirk” と “Dirk ” は同じです。日付の比較では、< は「より前」を意味し、> は「より後」を意味します。すべての char、nchar、unichar、unitext、varchar、nvarchar、univarchar、text、date/time データは、一重引用符か二重引用符で囲んでください。

```
select *
from titleauthor
where royaltyper < 50


select authors.au_lname, authors.au_fname
from authors
where au_lname > "McBadden"


select au_id, phone
from authors
where phone != "415 658-9932"


select title_id, newprice = price * 1.15
from  pubs2..titles
where advance > 5000
```

date/time データの入力については、「第 7 章 データの追加、変更、転送、削除」を参照してください。

not は式を否定します。否定の論理演算子 (not) と否定の比較演算子 (!>) では、位置が違うことに注意してください。

```
select title_id, type, advance
from titles
where (type = "business" or type = "psychology")
and not advance >5500

select title_id, type, advance
from titles
where (type = "business" or type = "psychology")
and advance !>5500
```

どちらも、同じ結果セットを返します。

```
title_id  type            advance
--------  ------------    --------
BU1032    business        5,000.00
BU1111    business        5,000.00
BU7832    business        5,000.00
PS2091    psychology      2,275.00
PS3333    psychology      2,000.00
PS7777    psychology      4,000.00

(6 rows affected)
```

範囲 (*between* および *not between*)

between キーワードでは、特定の範囲をすべて指定できます。

たとえば、売り上げが、4095 と 12,000 も含めたその間である本をすべて検索するには、次のクエリを使用します。

```
select title_id, total_sales
from titles
where total_sales between 4095 and 12000
```

```
title_id  total_sales
------    -----------
BU1032           4095
BU7832           4095
PC1035           8780
PC8888           4095
TC7777           4095

(5 rows affected)
```

より大きい (>) およびより小さい (<) 演算子を使用すると、上限値と下限値を含まない範囲を指定できます。

```
select title_id, total_sales
from titles
where total_sales > 4095 and total_sales < 12000
```

```
title_id  total_sales
------    -----------
PC1035           8780

(1 row affected)
```

not between は、指定された範囲外のすべてのローを検索します。売り上げが 4095 から 12,000 の範囲にない本をすべて検索するには、次を入力します。

```
select title_id, total_sales
from titles
where total_sales not between 4095 and 12000
```

```
title_id     total_sales
```

```
--------     -----------
BU1111              3876
BU2075             18722
MC2222              2032
MC3021             22246
PS1372               375
PS2091              2045
PS2106               111
PS3333              4072
PS7777              3336
TC3218               375
TC4203             15096

(11 rows affected)
```

リスト (*in*、*not in*)

in キーワードを使用すると、値リストのいずれかに一致する値を選択することができます。式には定数またはカラム名が指定でき、値リストには一組の定数またはサブクエリを指定できます。in キーワードに続く項目は、カンマで区切り、値リスト全体をカッコで囲みます。char、varchar、unichar、unitext、univarchar、datetime 値は、一重引用符か二重引用符で囲んでください。

たとえば、カリフォルニア州、インディアナ州、またはメリーランド州に住むすべての作家の名前と州のリストを表示するには、以下を使用します。

```
select au_lname, state
from authors
where state in ("CA", "IN", "MD")
 au_lname         state
 -----------      -----
 White            CA
 Green            CA
 Carson           CA
 O'Leary          CA
 Straight         CA
 Bennet           CA
 Dull             CA
 Gringlesby       CA
 Locksley         CA
 Yokomoto         CA
 DeFrance         IN
 Stringer         CA
 MacFeather       CA
 Karsen           CA
 Panteley         MD
 Hunter           CA
 McBadden         CA
```

クエリに in キーワードを使用すると、次の長いクエリと同じ結果セットが得られます。

```
select au_lname, state
from authors
where state = "CA" or state = "IN" or state = "MD"
```

in キーワードの最も重要な用途は、おそらく、「サブクエリ」とも呼ばれるネストされたクエリでの使用です。「第 5 章 サブクエリ：他のクエリ内でのクエリの使用」を参照してください。

たとえば、共著の本について、印税総額の 50 パーセントより少ない印税を受け取る作家の名前を調べるとします。authors テーブルには作家の名前が、titleauthor テーブルには印税の情報が格納されています。in を使用するが同じ from 句内にはリストしないで 2 つのテーブルを使用することによって、必要な情報を抽出できます。次のクエリで、その例を示します。

- titleauthor テーブルで、任意の本の印税を 50 パーセント未満受け取るすべての作家の au_id を検索します。
- authors テーブルから、titleauthor クエリの結果に一致する au_id を持つ作家の名前をすべて選択します。結果は、何人かの作家が 50 パーセント未満のカテゴリに当てはまることを示します。

```
select au_lname, au_fname
from authors
where au_id in
(select au_id
   from titleauthor
   where royaltyper < 50

au_lname          au_fname
--------------    ------------
Green             Marjorie
O'Leary           Michael
Gringlesby        Burt
Yokomoto          Akiko
MacFeather        Stearns
Ringer            Anne

(6 rows affected)
```

not in は、リストの項目に一致しない作家を検索します。次のクエリは、50% 未満の印税を受け取った本が 1 冊もない作家の名前を検索します。

```
select au_lname, au_fname
from authors
where au_id not in
(select au_id
   from titleauthor
   where royaltyper < 50

au_lname          au_fname
---------------   ------------
```

```
White               Johnson
Carson              Cheryl
Straight            Dick
Smith               Meander
Bennet              Abraham
Dull                Ann
Locksley            Chastity
Greene              Morningstar
Blotchet-Halls      Reginald
del Castillo        Innes
DeFrance            Michel
Stringer            Dirk
Karsen              Livia
Panteley            Sylvia
Hunter              Sheryl
McBadden            Heather
Ringer              Albert
Smith               Gabriella

(18 rows affected)
```

パターン一致

where 句内にワイルドカード文字を記述して、未知の文字を検索したり、共通の特徴に従ってデータをグループ化できます。この項では、SQL と Transact-SQL を使用したパターン一致について説明します。パターン一致の詳細については、『リファレンス・マニュアル：コマンド』を参照してください。

照合文字列：*like*

like キーワードは、パターンに一致する文字列を検索します。like は、char、varchar、nchar、nvarchar、unichar、unitext、univarchar binary、varbinary、text、および date/time データとともに使用されます。

like の構文は次のとおりです。

```
{where | having} [not]
    column_name [not] like  "match_string"
```

match_string には、表 2-2 の記号を含めることができます。

表 2-2: 照合文字列の特殊記号

記号	意味
%	0 文字以上の文字列と一致する
_	単一の文字と一致する

[*specifier*]	角カッコは、[a-f] または [abcdef] のように、範囲またはセットを囲む。*specifier* には次のように 2 種類の形式がある。
	• *rangespec1 – rangespec2*:
	  • *rangespec1* は文字の範囲の開始を示す。
	  • - は範囲を示す特殊文字。
	  • *rangespec2* は文字の範囲の終わりを示す。
	• *set*：任意の連続しない値が任意の順序で配列されたもの ([a2bR] など)。範囲 [a – f]、セット [abcdef]、セット [fcbdae] は同じ値のセットを返す。
	指定子は大文字と小文字を区別する。

[^*specifier*]	指定子の前に置かれる脱字記号 (^) は、範囲に含まないことを意味する。[^a-f] は「a-f の範囲にない」ことを意味し、[^a2bR] は「a、2、b、または R ではない」ことを意味する。

カラム・データを定数、変数、または表 2-2に示す「ワイルドカード」文字を含む他のカラムと一致させることができます。定数を使用するときは、照合文字列を引用符で囲みます。たとえば authors テーブルのデータとともに like を使用するとします。

- like “Mc%” は、Mc で始まる名前 (McBadden) をすべて検索します。
- like “%inger” は、inger という文字で終わる名前 (Ringer, Stringer) をすべて検索します。
- like “%en%” は、en という文字を含む名前 (Bennet, Green, McBadden) をすべて検索します。
- like “_heryl” は、heryl という文字で終わる 6 文字の名前 (Cheryl) をすべて検索します。
- like “[CK]ars[eo]n” は、Carsen,、Karsen、Carson、Karson (Carson) を検索します。
- like “[M-Z]inger” は、M から Z の間の任意の 1 文字で始まり、inger という文字で終わる名前 (Ringer) をすべて検索します。
- like “M[^c]%” は、“M” という文字で始まり、2 番目の文字に “c” を持たない名前をすべて検索します。

このクエリは、authors テーブルから市外局番が 415 であるすべての電話番号を検索します。

```
select phone
from authors
where phone like "415%"
```

text カラムに使用できる唯一の where 条件は、like です。このクエリは、blurbs テーブル内の、copy カラムに “computer” という語が含まれるすべてのローを検索します。

```
select * from blurbs
where copy like "%computer%"
```

Adaptive Server は、like を指定せずに使用されるワイルドカード文字を、パターンではなくリテラルとして解釈し、その値そのものを表すと解釈します。次のクエリは、4 文字の “415%” だけで構成される電話番号を検索しようとします。415 で開始する電話番号を検索するのではありません。

```
select phone
from authors
where phone = "415%"
```

like を datetime 値と一緒に使用すると、Adaptive Server は値を標準の datetime フォーマットに変換してから、varchar または univarchar に変換します。標準の格納フォーマットには秒やミリ秒は含まれていないため、like とパターンを使用して秒やミリ秒を検索することはできません。

エントリにさまざまな日付要素が含まれている可能性があるので、date and time 値を検索するときは like を使用します。たとえば、arrival_time という datetime カラムに “9:20” を挿入した場合、次のクエリは値を検索できません。Adaptive Server がこのエントリを “Jan 1 1900 9:20AM” に変換してしまうためです。

```
where arrival_time = "9:20"
```

ただし次のクエリは 9:20 という値を検索します。

```
where arrival_time like "%9:20%"
```

like トランザクションには date データ型と time データ型も使用できます。

*not like*の使用

not like では、like で使用できるものと同じワイルドカード文字を使用できます。たとえば、authors テーブル内の、市外局番が 415 でないすべての電話番号を検索するには、次のいずれかのクエリを使用します。

```
select phone
from authors
where phone not like "415%"
```

```
select phone
from authors
where not phone like "415%"
```

not like と ^ による異なる結果の取得

not like パターンは、必ずしも like と否定のワイルドカード文字 [^] で複製できるものではありません。否定のワイルドカード文字を持つ照合文字列は一度に 1 文字ずつ、段階を追って評価されます。評価のある時点で失敗した一致は、削除されます。

たとえば次のクエリは、データベース内の名前が “sys” で始まるシステム・テーブルを検索します。

```
select name
from sysobjects
where name like "sys%"
```

オブジェクトの合計数が 32 で、like でパターンが一致する名前が 13 個見つかったとすると、not like ではパターンに一致しないオブジェクトが 19 個見つかります。

```
where name not like "sys%"
```

一方、次のパターンでは異なる結果が得られることがあります。

```
like [^s][^y][^s]%
```

この場合、19 個ではなく、14 個だけが検出されることがあります。ここでは、“s” で始まる名前、“y” を 2 番目の文字として持つ名前、“s” を 3 番目の文字として持つ名前がすべて、システム・テーブル名と同様に検索結果から外されます。

リテラル文字としてのワイルドカード文字の使用

ワイルドカード文字をエスケープし、リテラルとして検索することによって、ワイルドカード文字を検索できます。like 照合文字列でワイルドカード文字をリテラルとして使用するには、2 つの方法があります。角カッコを使用する方法と escape 句を使用する方法です。照合文字列は、テーブル内のワイルドカード文字を含む値または変数としても使用できます。

角カッコ (Transact-SQL 拡張機能)

パーセント記号、アンダースコア、右と左の角カッコの文字には、角カッコを使用します。範囲を指定するためにダッシュを使用するのではなく、ダッシュを検索するには、角カッコ内の最初の文字としてダッシュを使用します。

表 2-3: 角カッコを使用したワイルドカード文字の検索

like 句	検索対象
like "5%"	後に 0 文字以上の文字列が続く 5
like "5[%]"	5%
like "_n"	an、in、on、など
like "[_]n"	_n
like "[a-cdf]"	a、b、c、d、または f
like "[-acdf]"	-、a、c、d、または f
like "[[ ]"	[
like "[ ]]"	]

escape 句 (SQL 準拠)

escape 句を使用して、like 句にエスケープ文字を指定します。エスケープ文字は 1 文字の文字列でなければなりません。サーバのデフォルト文字セット内の任意の文字を使用できます。

表 2-4: escape 句の使用

like 句	検索対象
like "5@%" escape "@"	5%
like "*_n" escape "*"	_n
like "%80@%%" escape "@"	80% を含む文字列
like "*_sql**%" escape "*"	_sql* を含む文字列
like "%#####_#%%" escape "#"	##_% を含む文字列

エスケープ文字は、指定された like 句内だけで有効であり、同じ文の他の like 句には影響しません。

エスケープ文字に後続する有効な文字は、ワイルドカード文字 (_、%、[、]、[^]) とエスケープ文字そのものだけです。エスケープ文字は、後続の 1 文字だけに影響します。パターンに、エスケープ文字である文字がリテラルとして 2 つ含まれている場合、文字列には連続する 4 つのエスケープ文字が含まれる必要があります (表 2-4 の最後の例を参照してください)。含まれていない場合は、SQLSTATE エラーステートが発生し、Adaptive Server はエラー・メッセージを返します。

複数のエスケープ文字を指定すると、SQLSTATE エラーステートが発生し、Adaptive Server はエラー・メッセージを返します。

```
like "%XX_%" escape "XX"
like "%XX%X_%" escape "XX"
```

ワイルドカード文字と角カッコの関係

エスケープ文字は、ワイルドカード文字とは異なり、角カッコ内でもその特別な意味を保持します。次のような理由から、既存のワイルドカード文字を escape 句内でエスケープ文字として使用しないでください。

- 角カッコまたはパーセント記号 (“_” または “%”) をエスケープ文字として指定すると、その like 句内での特別な意味を失い、エスケープ文字としてしか機能しません。
- 左または右の角カッコ (“[” と “]”) をエスケープ文字として指定すると、like 句内ではカッコの Transact-SQL 機能としての意味が無効になります。
- ハイフンまたは脱字記号 (“-” または “^”) をエスケープ文字として指定すると、通常の角カッコ内での特別な意味を失い、エスケープ文字としてしか機能しません。

後続ブランクと % の使用

like “% ” (2 つのスペースが続くパーセント記号) は “X ” (1 つのスペース)、“X ” (2 つのスペース)、“X ” (3 つのスペース)、または任意の後続スペースに一致します。

カラムでのワイルドカード文字の使用

ワイルドカード文字をカラムおよびカラム名に使用できます。特売の価格の射影を実行するために、pubs2 データベース内に special_discounts というテーブルを作成するとします。

```
create table special_discounts
id_type char(3), discount int)
insert into special_discounts
values("BU%", 10)
...
```

テーブルには、次のデータが含まれています。

```
id_type discount
------- -----------
BU%              10
PS%              12
MC%              15
```

次のクエリは where 句内の id_type にワイルドカード文字を使用します。

```
select title_id, discount, price, price - (price*discount/100)
from special_discounts, titles
where title_id like id_type
```

クエリの結果は、次のようになります。

```
title_id   discount      price

 -------- ----------- -------------- --------------
 BU1032            10          19.99          17.99
 BU1111            10          11.95          10.76
 BU2075            10           2.99           2.69
 BU7832            10          19.99          17.99
 PS1372            12          21.59          19.00
 PS2091            12          10.95           9.64
 PS2106            12           7.00           6.16
 PS3333            12          19.99          17.59
 PS7777            12           7.99           7.03
 MC2222            15          19.99          16.99
 MC3021            15           2.99           2.54
 MC3026            15           NULL           NULL

(12 rows affected)
```

この種の例では、一連の or 句を構成することなく、高度なパターン一致が可能です。

「不定の値」：null

カラム内の null は、そのカラムに入力が行われていないことを意味します。カラムのデータ値は「不定」または「未知」です。

null は「0」や「ブランク」と同義ではありません。null 値を使用すると、数値カラムの 0 や文字カラムのブランクなどの意図的な入力と、入力が行われていないことの識別が可能になります。入力が行われていない数値カラムと文字カラムは、いずれも null になります。

null は、null 値が許可されているカラムに次の 2 つの方法で入力できます。

- データを入力しないと、Adaptive Server は自動的に “null” を入力します。
- ユーザは、引用符を付けずに “NULL” または “null” と明示的に入力できます。

“null” を引用符付きで文字カラムに入力すると、null 値ではなくデータとして扱われます。

クエリ結果では、null という単語が表示されます。たとえば titles テーブルの advance カラムは、null 値を許可します。そのカラムのデータを調べることによって、契約による前払い金がない (MC2222 のローで advance カラムが 0) か、前払い金の額がデータ入力時に不定であった (MC3026 で advance カラムが null) かを識別できます。

```
select title_id, type, advance
from titles
where pub_id = "0877"

title_id     type             advance
--------     ----------       ---------
```

```
MC2222       mod_cook             0.00
MC3021       mod_cook        15,000.00
MC3026       UNDECIDED            NULL
PS1372       psychology       7,000.00
TC3218       trad_cook        7,000.00
TC4203       trad_cook        4,000.00
TC7777       trad_cook        8,000.00

(7 rows affected)
```

null 値のあるカラムのテスト

where、if、while 句に is null を使用して (「第 15 章 バッチおよびフロー制御言語の使用」を参照)、カラム値を null と比較し、比較の結果に基づいて選択を行ったり特定のアクションを実行したりします。true の値を返すカラムだけが、選択されるかまたは指定のアクションを実行します。false や unknown を返すカラムは行いません。

次の例は、advance が 5000 未満または null となっているローだけを選択します。

```
select title_id, advance
from titles
where advance < 5000 or advance is null
```

Adaptive Server は、使用する「演算子」と比較を行う値の種類によって、null 値を異なる方法で扱います。一般に、null の値を比較した結果は、unknown になります。これは、null が特定の値や他の null に等しいか (または等しくないか) どうかを判断できないためです。次のような場合には、*expression* が、null と評価されるカラム、変数、リテラル、またはその組み合わせであると、true が返されます。

- *expression* is null
- *expression* = null
- *expression* = *@x* (この *@x* は null を含む変数またはパラメータです。この例外によって、null のデフォルト・パラメータを使用したストアド・プロシージャの作成が簡単になります)。
- *expression* != *n*。この *n* は null を含まないリテラルで、*expression* は null と評価されます。

これらの式の否定は、式が null と評価されないときに true を返します。

- *expression* is not null
- *expression* != null
- *expression* != *@x*

キーワード like および not like を演算子 = および != の代わりに使用すると、反対になります。次の比較では true が返されます。

- *expression* not like null

次の比較では false が返されます。

- *expression* like null

これらの式の右端の値は、リテラルの null であるか、null を含む変数またはパラメータです。比較の一番右側が式 (*@nullvar* + 1 など) の場合、式全体が null と評価されます。

null カラム値は他の null カラム値とジョインしません。1 つの where 句の中で、null のカラムの値を他の null のカラムの値と比較すると、どの比較演算子を使った場合にも、null の値には unknown が返され、結果にそのローは含まれません。たとえば次のクエリは、両方のテーブルにおいて column1 に null が含まれている場合、ローを返しません (ただし他のローを返すことはあります)。

```
select column1
from table1, table2
where table1.column1 = table2.column1
```

次の演算子は、null とともに使用された場合に結果を返します。

- = は null を含むすべてのローを返す。
- != または <> は null を含まないすべてのローを返す。

SQL 規格に準拠するために set ansinull が “on” に設定されている場合、= および != 演算子は null とともに使用しても結果を返しません。set ansinull オプション値に関係なく、<、<=、!<、>、>=、!> 演算子は null とともに使用される場合は値を返しません。

Adaptive Server はカラム値が null であると判断します。そのため、これは true であるとみなされます。

```
column1 =  NULL
```

ただし、null は「不定の値を持っている」という意味なので、次の比較は判別できません。

```
where column1 > null
```

2 つの不定の値が同じであると仮定することはできません。

この論理は、where 句内で 2 つのカラム名を使用するとき、つまり 2 つのテーブルをジョインするときにも適用されます。“where column1 = column2” のような句は、カラムに null 値が含まれているローを返しません。

次のパターンを使用して null 値または非 null 値を検索することもできます。

```
where column_name is [not] null
```

次に例を示します。

```
where advance < 5000 or advance is null
```

titles テーブルのローの中には、不完全なデータを含むものがあります。たとえば、『The Psychology of Computer Cooking』(title_id = MC3026) という本が企画され、そのタイトル、タイトル ID 番号、候補となる出版社が入力されたとします。しかし作家とはまだ契約しておらず、詳細が未定であるため、price、advance、royalty、total_sales、notes カラムには null 値が表示されています。null 値は比較のいずれにも一致しないので、前払い金が 5000 未満である本のすべてのタイトル ID 番号と前払い金を求めるクエリでは、『The Psychology of Computer Cooking』を検索できません。

```
select title_id, advance
from titles
where advance < 5000
```

```
title_id  advance
--------  ----------
MC2222         0.00
PS2091     2,275.00
PS3333     2,000.00
PS7777     4,000.00
TC4203     4,000.00

(5 rows affected)
```

advance カラムの前払い金が 5000 ドル未満、または null の本を求めるクエリを次に示します。

```
select title_id, advance
from titles
where advance < 5000
  or advance is null
```

```
title_id  advance
--------  ----------
MC2222         0.00
MC3026         NULL
PC9999         NULL
PS2091     2,275.00
PS3333     2,000.00
PS7777     4,000.00
TC4203     4,000.00

(7 rows affected)
```

create table 文の null、および null とデフォルトの関係の詳細については、「第 8 章 データベースおよびテーブルの作成」を参照してください。テーブルへの null 値の挿入の詳細については、「第 7 章 データの追加、変更、転送、削除」を参照してください。

false と unknown の違い

false も unknown には論理的に重要な違いがあり、false の反対 (“not false”) は true ですが、unknown の反対はそのまま unknown です。たとえば、“1 = 2” は false と評価され、その反対の “1 != 2” は true と評価されます。しかし、“not unknown” は、unknown のままです。比較に null 値が含まれている場合は、式を否定して反対のローの集合や反対の真理値を取得することはできません。

null と値との置き換え

isnull 組み込み関数を使用して、null を特定の値に置き換えることができます。置き換えは表示目的のみに行われます。実際のカラム値には影響はありません。構文は次のとおりです。

isnull(*expression*, *value*)

たとえば、titles からすべてのローを選択し、カラム notes の null 値を unknown という値で表示するには、次の文を使用します。

```
select isnull(notes, "unknown")
from titles
```

null に評価される式

算術演算子またはビット処理演算子を持つ式は、いずれかのオペランドが null である場合は、null と評価されます。次の式は column1 が null であれば null と評価されます。

```
1 + column1
```

文字列と null の連結

文字列と null を連結すると、式は文字列に評価されます。次に例を示します。

```
select "abc" + NULL + "def"

-----
abcdef
```

システム生成の null

Transact-SQL では、convert のようなシステム関数の結果から生成されるシステム生成の null と、ユーザが割り当てた null とは動作が異なります。たとえば次の文では、ユーザ指定の null と 1 との不等関係比較では true を返します。

```
if (1 != NULL) print "yes" else print "no"

yes
```

システム生成の null との同様の比較では unknown が返されます。

```
if (1 != convert(integer, NULL))
print "yes" else print "no"

no
```

より一貫した動作にするには、set ansinull を有効にします (on に設定)。こうすると、システム生成の null でもユーザ提供の null でも、比較では unknown が返されるようになります。

論理演算子による条件の結合

「論理演算子」である and、or、および not は、where 句の探索条件を結合します。構文は次のとおりです。

```
{where | having} [not]
    column_name join_operator column_name
```

join_operator は比較演算子で、*column_name* は比較に使用されるカラムです。カラムの名前があいまいな場合は、名前を修飾します。

and は 2 つ以上の条件を結合して、すべての条件が true の場合にだけ、結果を返します。たとえば次のクエリは、作家の姓が Ringer で名前が Anne のローだけを検索します。Albert Ringer のローは検索しません。

```
select *
from authors
where au_lname = "Ringer" and au_fname = "Anne"
```

or も 2 つ以上の条件を結合しますが、条件のいずれかが true の場合に結果を返します。次のクエリは、au_fname カラムに Anne か Ann を含むローを検索します。

```
select *
from authors
where au_fname = "Anne" or au_fname = "Ann"
```

and および or 条件は 252 個まで指定できます。

not は、後続の式を否定します。次のクエリは、カリフォルニア州に住まないすべての作家を選択します。

```
select * from authors
where not state = "CA"
```

1 つの文で複数の論理演算子を使用する場合、通常は and 演算子の後に or 演算子が評価されます。実行の順序は括弧を使用して変更できます。次に例を示します。

```
select * from authors
where (city = "Oakland" or city = "Berkeley") and state = "CA"
```

論理演算子の優先度

算術およびビット処理演算子は論理演算子の前に処理されます。1 つの文で複数の論理演算子を使用する場合、not が最初に評価され、次に and、最後に or が評価されます。「ビット処理演算子」(18 ページ) を参照してください。

たとえば次のクエリは、前払い金が 5500 を超える心理学の本と、前払い金に関係なく titles テーブルのすべてのビジネス関連の本を検索します。and は or の前に処理されるため、前払い金の条件は心理学の本のみを対象とします。

```
select title_id, type, advance
from titles
where type = "business" or type = "psychology"
  and advance > 5500

title_id  type        advance
--------  ----------  ----------
BU1032    business     5,000.00
BU1111    business     5,000.00
BU2075    business    10,125.00
BU7832    business     5,000.00
PS1372    psychology   7,000.00
PS2106    psychology   6,000.00

(6 rows affected)
```

カッコを追加して、or の評価を先に実行するよう、クエリの意味を変更できます。次のクエリは、前払い金が 5500 を超えるすべてのビジネス書および心理学の本を検索します。

```
select title_id, type, advance
from titles
where (type = "business" or type = "psychology")
  and advance > 5500

title_id  type         advance
--------  ----------   ---------
BU2075    business     10,125.00
PS1372    psychology    7,000.00
PS2106    psychology    6,000.00

(3 rows affected)
```

ネストされた exists クエリでの複数の select 項目の使用

次の例における複数カラムの使用は、ネストされた exists クエリで 1 つの c1 または c2 カラムを選択する場合と同じです。

```
1> create table t1(c1 int, c2 int)
2> go
1> create table t2(c1 int, c2 int)
2> go
1> select * from t1 where exists (select c1, c2
   from t2)
2> go
```

次のようにアスタリスクを他の select 項目と混在させることはできません。

```
1> select * from t1
    where exists (select t2.*, c1 from t2)
2> go

Msg 102, Level 15, State 1:
Line 1:
Incorrect syntax near ','.
```

ネストされた *select* 文でのカラムのエイリアスの使用

ネストされた select 文の select リストには、カラムのエイリアスを使用できます。カラムのエイリアスには、次のいずれかの形式を使用します。

- column_heading = *expression*
- *expression* column_heading
- *expression* as column_heading

たとえば、次の例は as tableid 句を削除した select 文に相当します。

```
1> select *
2> from syscolumns c
3> where c.id in (
4> select o.id as tableid
5> from sysobjects o
6> where o.name like '%attr%')
```

この例では、Adaptive Server によりエイリアス (許可されたカラム見出し) が無視されます。

第 3 章

集合、グループ化、ソートの使用

この章では、クエリで取得したデータを計算するための集合関数 sum、avg、count、count(*)、count_big、count_big(*)、max、min について取り上げます。group by 句、having 句、order by 句を使用してデータをカテゴリおよびサブグループに編成する方法について説明します。また、compute 句と union 演算子の 2 つの Transact-SQL 拡張機能についても説明します。

使用する Adaptive Server が大文字と小文字を区別しない場合は、返されるデータに大文字と小文字の区別がどのように影響するかについて、『リファレンス・マニュアル：コマンド』の「group by 句と having 句」および「compute 句」の項を参照してください。

集合関数の使用

集合関数は sum、avg、count、min、max、count_big、count(*)、count_big(*) です。また、「集合関数」を使用して、データを計算してまとめることができます。たとえば、pubs2 データベース内の titles テーブルで販売された本の数を確認するには、次のように入力します。

```
select sum(total_sales
from titles

-------------
       97746
```

例では集合カラム用のカラム見出しはありません。

集合関数は、値を処理するカラム名を引数として使用します。集合関数は、テーブル内のすべてのロー、where 句で指定したテーブルのサブセット、またはテーブル内のローの 1 つ以上のグループに適用できます。Adaptive Server は、集合関数が適用されたロー・セットごとに 1 つの値を生成します。

集合関数の構文を次に示します。

aggregate_function ([all | distinct] *expression*)

expression は通常はカラム名です。ただし、定数、関数、または算術演算子かビット処理演算子で連結されたカラム名、定数、関数の任意の組み合わせにすることも可能です。case 式またはサブクエリも式中に使用できます。

たとえば次の文を使用して、価格が 2 倍になることを仮定した場合の、すべての本による平均価格を計算できます。

```
select avg(price * 2)
from titles

-------------
        29.53

(1 row affected)
```

重複する値を削除してから集合関数を適用するには、sum、avg、count、min、および max とともにオプションのキーワード distinct を使用します。すべてのローに操作を実行する all はデフォルトです。

表 3-1: 集合関数の構文と結果

集合関数	結果
sum([all | distinct] *expression*)	式中の (重複しない) 値の合計
avg([all | distinct] *expression*)	式中の (重複しない) 値の平均
count([all | distinct] *expression*)	integer として返された、式中の (重複しない) null 以外の値の数
count_big ([all | distinct] *expression*)	bigint として返された、式中の (重複しない) null 以外の値の数
count(*)	integer として選択されたローの数
count_big(*)	bigint として選択されたローの数
max(*expression*)	式中の最も高い値
min(*expression*)	式中の最も低い値

集合関数は、前述の例のように select リスト内で使用したり、having 句内で使用できます。having 句については、「データのグループの選択：having 句」(86 ページ) を参照してください。

where 句内では集合関数を使用できませんが、select リスト内に集合関数を持つほとんどの select 文には、集合が適用されるローを制限する where 句が含まれます。これより前の項で示した例では、各集合関数は、テーブル全体について単一の計算値を生成しました。

select 文に where 句が含まれているが group by 句は含まれていない場合 (「クエリ結果のグループ構成：group by 句」(75 ページ)を参照)、集合関数はローのサブセットについて「スカラ集合」という単独の値を生成します。ただし、select 文では結果テーブルのそれぞれローについて単一の値を繰り返すカラムを select リストに含めることができます (Transact-SQL 拡張機能)。

このクエリは、ビジネス関連の本だけについての、前払い金の平均と売り上げ高を返し、“advance and sales” というカラム名を前に付けます。

```
select "advance and sales", avg(advance), sum(total_sales)
from titles
where type = "business"

-----------------  ----------------- ------------
advance and sales          6,281.25       30788

(1 row affected)
```

集合関数とデータ型

集合関数は、次に示す例外を除く任意の型のカラムで使用できます。

- sum と avg は bigint、int、smallint、tinyint、unsigned bigint、unsigned int、unsigned smallint、decimal、numeric、float、money の数値カラムにだけ使用できます。
- min および max は bit データ型には使用できません。
- count(*) と count_big(*) 以外の集合関数は text および image データ型には使用できません。

たとえば、min (minimum) を使用して、文字型カラムの最低値 (アルファベットの初めの方に最も近い値) を検出できます。

```
select min(au_lname)
from authors

--------------------------
Bennet

(1 row affected)
```

ただし、テキスト・カラムの内容を平均することはできません。

```
select avg(au_lname)
from authors

Msg 257, Level 16, State 1:
--------------------------
(1 row affected)
Line 1:
Implicit conversion from datatype 'VARCHAR' to 'INT' is not
allowed.Use the CONVERT function to run this query.
```

count と *count(*)*

count では式内の null 以外の値を持つローの数を検出しますが、count(*) ではテーブル内のローの総数を検出します。次の文は本の総数を検出します。

```
select count(*)
from titles

------------------
                18

(1 row affected)
```

count(*) は指定のテーブル内のローの数を返します。重複は削除しません。null 値を持つローを含め、各ローをカウントします。

他の集合関数と同様、count(*) を、select リスト内の別の集合や where 句などと組み合わせることができます。

```
select count(*), avg(price) from titles
where advance > 1000

---------- ---------
        15     14.42

(1 row affected)
```

distinct を使った集合関数

オプションのキーワード distinct を使用できるのは、sum、avg、count_big、および count のみです。distinct を使用すると、Adaptive Server は重複する値を削除してから、計算します。

distinct を使用する場合は、引数に算術式を含めることはできません。引数はカラム名だけを使用します。distinct はカッコで囲み、カラム名の前に置きます。たとえば、作家が住む都市の数を、重複を計算せずに検出するには、次のように入力します。

```
select count(distinct city)
from authors

-------------
           16

(1 row affected)
```

ビジネス関連の本すべてによる平均価格を厳密に計算するには、distinct を省略します。次の文は、ビジネス関連の本すべてによる平均価格を返します。

```
select avg(price)
from titles
where type = "business"

-------------
```

```
        13.73

(1 row affected)
```

ただし、2 冊以上の本の価格が同じ場合、distinct を使用すると、共通の価格は一度しか計算に含まれません。

```
select avg(distinct price)
from titles
where type = "business"

-------------
        11.64

(1 row affected)
```

null 値と集合関数

Adaptive Server は、集合関数が演算を実行しているカラム内の null 値を無視します (count(*) と count_big(*) を除く)。ansinull を on に設定すると、Adaptive Server は null 値が無視されるたびにエラー・メッセージを返します。『リファレンス・マニュアル：コマンド』を参照してください。

たとえば、titles テーブルの前払い金の count はタイトル名の count とは異なります。これは advance カラムに null 値があるためです。

```
select count(advance)
from titles

-------------
           16

(1 row affected)

select count(title)
from titles

-------------
           18

(1 row affected)
```

カラム内のすべての値が null の場合、count は 0 を返します。where 句で指定した条件を満たすローがない場合、count は 0 を返します。その他の関数はすべて null を返します。例を示します。

```
select count(distinct title)
from titles
where type = "poetry"

-------------
            0
```

```
(1 row affected)

select avg(advance)
from titles
where type = "poetry"

-------------
          (NULL)

(1 row affected)
```

統計集合の使用

集合関数は、データベースに含まれるローのグループのデータを要約します。sum、avg、max、min、count_big、count などの単純な集合関数は、select リスト、having、order by 句、そして select 文の compute 句のみで許可されます。これらの関数は、データベースに含まれるローのグループのデータを要約します。

Adaptive Server は、数値データの統計的分析を行うための統計集合関数をサポートするようになりました。統計集合関数には、stddev、stddev_samp、stddev_pop、variance、var_samp、var_pop が含まれます。『リファレンス・マニュアル：ビルディング・ブロック』を参照してください。

stddev と variance を含むこれらの関数は、クエリの group by 句の指定に従ってローのグループの値を計算できる集合関数です。max や min などのその他の基本的な集合関数と同様に、これらの計算は入力データ内の null 値を無視します。分散と標準偏差の計算では必ず IEEE の倍精度浮動小数点数が使用されます。

分散関数または標準偏差関数への入力が空のデータ・セットである場合、これらの集合関数は結果として null 値を返します。分散関数または標準偏差関数への入力が単一の値である場合、これらの関数は結果として 0 を返します。

標準偏差と分散

新しい統計集合関数 (とそのエイリアス) は、次のとおりです。

- stddev_pop (同様に stdevp) − 母標準偏差。グループの各ロー (distinct が指定されている場合、重複が削除された後に残る各ロー) に対して評価される、指定された値式の母標準偏差を計算します。これは、母分散の平方根として定義されます。
- stddev_samp (同様に stdev と stddev) − 標本標準偏差。グループの各ロー (distinct が指定されている場合、重複が削除された後に残る各ロー) に対して評価される、指定された値式の母標準偏差を計算します。これは、標本分散の平方根として定義されます。

- var_pop (同様に varp) － 母分散。グループの各ロー (distinct が指定されている場合、重複が削除された後に残る各ロー) に対して評価される、指定された値式の母分散を計算します。これは、値式と値式の平均の差の 2 乗和を、グループ内のローの数で割った値として定義されます。
- var_samp (同様に var と variance) － 標本分散。グループの各ロー (distinct が指定されている場合は、重複が削除された後に残る各ロー) に対して評価される値式の標本分散を計算します。これは、値式と値式の平均の差の 2 乗和を、グループ内のローの数より 1 少ない数で割った値として定義されます。

『リファレンス・マニュアル：ビルディング・ブロック』を参照してください。

クエリ結果のグループ構成：*group by* 句

group by 句は、クエリの出力をいくつかのグループにします。1 つ以上のカラム名でグループ化したり、式中に数値データ型を使用して計算カラムの結果でグループ化することもできます。group by は集合関数とともに使用されると、サブグループごとの計算結果を取り出し、複数のローを返すことがあります。

group by カラム (または式) の最大数は明示的に制限されていません。group by の結果の唯一の制限は、group by カラムの幅と集約結果の合計が 64K を超えることができないことです。

注意　text、unitext、または image データ型のカラムでは group by を使用できません。

---

group by を集合関数なしで使用することもできますが、そのような使用方法では機能を限定し、混乱を招く結果を生成することがあります。次の例では、タイトルの種類によって結果をグループ分けします。

```
select type, advance
     from titles
group by type

type              advance
------------         ---------
popular comp          7,000.00
popular comp          8,000.00
popular comp              NULL
business              5,000.00
business              5,000.00
business             10,125.00
mod_cook                  0.00
mod_cook             15,000.00
trad_cook             7,000.00
trad_cook             4,000.00
```

```
trad_cook               8,000.00
UNDECIDED                   NULL
psychology              7,000.00
psychology              2,275.00
psychology              6,000.00
psychology              2,000.00
psychology              4,000.00

(18 rows affected)
```

advance カラムの集合がある場合、クエリは各グループに対して合計を返します。

```
select type, sum(advance)
   from titles
group by type
```

```
type
------------ ------------------------
popular_comp               15,000.00
business                   25,125.00
mod_cook                   15,000.00
trad_cook                  19,000.00
UNDECIDED                       NULL
psychology                 21,275.00

(6 rows affected)
```

集合を使用した group by 句内の計算値は、「ベクトル集合」と呼ばれ、1 つのローのみが返されるスカラ集合と異なります (「集合関数の使用」(69 ページ)を参照してください)。

『リファレンス・マニュアル：コマンド』を参照してください。

group by と SQL 規格

group by の SQL 規格は、Sybase の規格より厳密です。SQL 規格では、次のことを要求しています。

- select リストのカラムは、group by 式内でも指定されているか、または集合関数の引数となっている必要があります。
- group by 式は select リスト内にあって、ベクトル集合関数の引数として使用されないカラム名だけを指定できます。

複数の Transact-SQL 拡張機能 (次の項で説明) を使用して、これらの制限を緩和できます。ただし、複雑な結果セットは理解するのが困難になる可能性があります。次のように fipsflagger オプションを設定すると、Transact-SQL 拡張機能が使用されていることを示す警告メッセージが表示されます。

```
set fipsflagger on
```

fipsflagger オプションの詳細については、『リファレンス・マニュアル：コマンド』の「set コマンド」を参照してください。

group by を使用したグループのネスト

group by 句内に複数のカラムを含めてグループをネストします。group by で集合が確立されると、集合関数が適用されます。この文は、本の平均価格と売り上げの合計を、まず出版社の ID 番号でグループ化し、次に種類別にグループ化して算出します。

```
select pub_id, type, avg(price), sum(total_sales)
from titles
group by pub_id, type
pub_id type
------ ------------ ------ -------
0736 business 2.99 18,722
0736 psychology 11.48 9,564
0877 UNDECIDED NULL NULL
0877 mod_cook 11.49 24,278
0877 psychology 21.59 375
0877 trad_cook 15.96 19,566
1389 business 17.31 12,066
1389 popular_comp 21.48 12,875

(8 rows affected)
```

グループ内に他のグループをネストできます。group by カラム (または式) の最大数は明示的に制限されていません。

group by を使用したクエリ内の他のカラムの参照

SQL 規格は、group by 句に select リストの項目を含めることを要求しています。しかし、Transact-SQL では、集合関数を使用する場合もそうでない場合も、また group by と select リストのどちらにも、有効な任意のカラム名を指定できます。

次に示す拡張機能によって、Sybase は group by を持つクエリの select リストに指定または省略できる内容について制限をなくしました。

- select リストのカラムはグループ化カラムおよびベクトル集合に使用されるカラムに限定されない。
- group by で指定するカラムは、select リスト内の非集合カラムに限定されない。

ベクトル集合は、group by 句とともに使用する必要があります。SQL 規格は、select リストの非集合カラムが group by カラムと一致することを必要とします。ただし、前述の最初の項にあるように、クエリの select リストに拡張カラムを指定できます。

たとえば、SQL の多くのバージョンでは、拡張 title_id カラムを select リストに含めることは許可されていませんが、Transact-SQL ではこれを許可します。

```
select type, title_id, avg(price), avg(advance)
from titles
group by type
type title_id
------------ -------- ----- -------
business BU1032 13.73 6,281.25
business BU1111 13.73 6,281.25
business BU2075 13.73 6,281.25
business BU7832 13.73 6,281.25
mod_cook MC2222 11.49 7,500.00
mod_cook MC3021 11.49 7,500.00
UNDECIDED MC3026 NULL NULL
popular_comp PC1035 21.48 7,500.00
popular_comp PC8888 21.48 7,500.00
popular_comp PC9999 21.48 7,500.00
psychology PS1372 13.50 4,255.00
psychology PS2091 13.50 4,255.00
psychology PS2106 13.50 4,255.00
psychology PS3333 13.50 4,255.00
psychology PS7777 13.50 4,255.00
trad_cook TC3218 15.96 6,333.33
trad_cook TC4203 15.96 6,333.33
trad_cook TC7777 15.96 6,333.33
(18 rows affected)
```

前述の例では price および advance カラムを type カラムに基づいて集約していますが、その結果では各グループに含まれる本の title_id も表示しています。

前述の 2 番目の箇条書き項目に示すように、クエリの select リストのカラムとして指定されていないカラムもグループ化できます。カラムは結果には表示されませんが、ベクトル集合はその計算値を計算します。次に例を示します。

```
select state, count(au_id)
from authors
group by state, city

state
----- --------
CA 2
CA 1
CA 5
CA 5
CA 2
CA 1
CA 1
CA 1
CA 1
IN 1
KS 1
```

```
MD 1
MI 1
OR 1
TN 1
UT 2
(16 rows affected)
```

この例では、各グループにどの都市が属するかは表示されませんが、state と city の両方でベクトル集合の結果をグループ化しています。したがって、結果は誤解を招く可能性があります。

次に示すクエリが、前述のクエリに似た結果を生成すると考えることがあります。ベクトル集合だけが各ローについて各都市の数を計算すると考えられるためです。

```
select state, count(au_id)
from authors
group by city
```

しかし、結果は大幅に異なります。state および city カラムの両方に group by を使用しない場合、クエリは各都市の数は計算しますが、それを都市ごとに 1 つの結果ローにグループ化するのではなく、authors にあるその都市の各ローについての計算を表示します。

```
state
----- -----------
CA 1
CA 5
CA 2
CA 1
CA 5
KS 1
CA 2
CA 2
CA 1
CA 1
TN 1
OR 1
CA 1
MI 1
IN 1
CA 5
CA 5
CA 5
MD 1
CA 2
CA 1
UT 2
UT 2

(23 rows affected)
```

where 句やジョインを含む複雑なクエリで Transact-SQL 拡張機能を使用すると、結果はさらに理解が難しいものになる可能性があります。group by の結果が混乱や誤解を招かないようにするには、fipsflagger オプションを使用して Transact-SQL 拡張機能を含むクエリを識別することをおすすめします。詳細については、「group by と SQL 規格」(76 ページ) および『リファレンス・マニュアル：コマンド』を参照してください。

式と *group by*

もう 1 つの Transact-SQL 拡張機能を使用すると、集合関数を含まない式でグループ分けできます。次に例を示します。

```
select avg(total_sales), total_sales * price
from titles
group by total_sales * price

---------      -------------
     2045          22,392.75
     2032          40,619.68
     4072          81,399.28
     NULL               NULL
     4095          61,384.05
    18722          55,978.78
      375           7,856.25
    15096         180,397.20
     3876          46,318.20
      111             777.00
     3336          26,654.64
     4095          81,859.05
    22246          66,515.54
     8780         201,501.00
      375           8,096.25
     4095          81,900.00

(16 rows affected)
```

式“total_sales * price” が許可されます。

select リストではカラム見出し (「エイリアス」とも呼ばれます) を使用できますが、group by にそれを使用することはできません。次の文はエラー・メッセージを生成します。

```
select Category = type, title_id, avg(price), avg(advance)
from titles
group by Category
------------------
Msg 207, Level 16, State 4:
Line 1:
Invalid column name 'Category'
Msg 207, Level 16, State 4:
Line 1:
```

```
Invalid column name 'Category'
```

group by 句は “group by Category” ではなく、“group by type” とする必要があります。

```
select Category = type, title_id, avg(price), avg(advance)
from titles
group by type
--------------
         21.48
         13.73
         11.49
         15.96
           (NULL)
13.50

(6 rows affected)
```

ネストされた集合内での *group by* の使用

Transact-SQL 拡張機能により、スカラ集合内でベクトル集合をネストすることもできます。たとえば、非ネスト集合を使用してすべての種類の本の平均価格を調べるには、次のように入力します。

```
select avg(price)
from titles
group by type

---------------
(NULL)
13.73
11.49
21.48
13.50
15.96

(6 rows affected)
```

max 関数内に平均価格をネストすることによって、種類でグループ分けした本のグループの最も高い平均価格を算出できます。

```
select max(avg(price))
from titles
group by type

-------------
        21.48

(1 row affected)
```

定義により、group by 句は最も内側の集合に適用されます。この場合は avg になります。

null 値と *group by*

グループ化するカラムに null 値が 1 つ含まれる場合、そのローは結果内でそれ独自のグループになります。グループ化するカラムに複数の null 値が含まれる場合、その null 値は 1 つのグループを形成します。この例では group by と advance カラムを使用します。このカラムには null 値が含まれます。

```
select advance, avg(price * 2)
from titles
group by advance
```

```
advance
------------------  ------------------
              NULL                NULL
              0.00               39.98
           2000.00               39.98
           2275.00               21.90
           4000.00               19.94
           5000.00               34.62
           6000.00               14.00
           7000.00               43.66
           8000.00               34.99
          10125.00                5.98
          15000.00                5.98

(11 rows affected)
```

count(*column_name*) 集合関数を使用している場合、null 値を含むカラムでグループ化すると、グループ化されたローについては 0 のカウントが返されます。これは、count(*column_name*) が null 値を含まないためです。通常は、count(*) を使用します。この例では、null 値を含む titles テーブルからの price カラムについてグループ化とカウントを行い、比較のために count(*) を示します。

```
select price, count(price), count(*)
from titles
group by price
```

```
price
------------- ----- -----
        NULL     0     2
        2.99     2     2
        7.00     1     1
        7.99     1     1
       10.95     1     1
       11.95     2     2
       14.99     1     1
       19.99     4     4
       20.00     1     1
       20.95     1     1
       21.59     1     1
       22.95     1     1

(12 rows affected)
```

where 句と *group by*

group by のある文では、where 句を使用できます。where 句の条件を満たさないローは、グループ化が行われる前に除外されます。

```
select type, avg(price)
from titles
where advance > 5000
group by type

type
-------------    --------
business            2.99
mod_cook            2.99
popular_comp       21.48
psychology         14.30
trad_cook          17.97

(5 rows affected)
```

5000 ドルを超える前払い金があるローだけが、クエリ結果の生成に使用されるグループに含まれます。

しかし、Adaptive Server が select リスト内の余分のカラムおよび where 句を処理する方法は矛盾するように見えることがあります。次に例を示します。

```
select type, advance, avg(price)
from titles
where advance > 5000
group by type

type           advance
-------------  ---------  --------
business        5,000.00      2.99
business        5,000.00      2.99
business       10,125.00      2.99
business        5,000.00      2.99
mod_cook            0.00      2.99
mod_cook       15,000.00      2.99
popular_comp    7,000.00     21.48
popular_comp    8,000.00     21.48
popular_comp        NULL     21.48
psychology      7,000.00     14.30
psychology      2,275.00     14.30
psychology      6,000.00     14.30
psychology      2,000.00     14.30
psychology      4,000.00     14.30
trad_cook       7,000.00     17.97
trad_cook       4,000.00     17.97
trad_cook       8,000.00     17.97

(17 rows affected)
```

advance (拡張) カラムの結果を見ると、クエリが where 句を無視しているように見えるだけです。Adaptive Server は where 句を満たすローだけを使用してベクトル集合を計算しますが、select リストに含んだ拡張カラムのすべてのローを表示します。このような結果のローをさらに制限するには、having 句を使用します。「データのグループの選択：having 句」(86 ページ) を参照してください。

group by と all

group by 句の all キーワードは、Transact-SQL 拡張機能です。これは、キーワードを使用する select 文にも where 句が含まれている場合にだけ意味があります。

all を使用すると、探索条件を満たすローを持たないグループがある場合でも、クエリ結果には group by 句によって生成されたすべてのグループが含まれます。all を使用しないと、group by を含む select 文は、条件を満たすローがないグループを表示しません。

次に例を示します。

```
select type, avg(advance)
from titles
where advance > 1000 and advance < 10000
group by type

type
------------  ------------------------
business                     5,000.00
popular_comp                 7,500.00
psychology                   4,255.00
trad_cook                    6,333.33

(4 rows affected)

select type, avg(advance)
from titles
where advance > 1000 and advance < 10000
group by all type

type
------------  ------------------------
UNDECIDED                        NULL
business                     5,000.00
mod_cook                         NULL
popular_comp                 7,500.00
psychology                   4,255.00
trad_cook                    6,333.33

(6 rows affected)
```

最初の文は、1000 ドルを超えるが 10,000 ドル未満の前払い金を要する本についてのみ、グループを生成します。モダン・クッキングの本の前払い金はこの範囲にないため、mod_cook の結果にはグループはありません。

2 番目の文では、モダン・クッキングのグループには where 句で指定された条件に該当するローは含まれていなくても、モダン・クッキングと "UNDECIDED" を含むすべての種類のグループを生成します。Adaptive Server は、修飾ローを持たないすべてのグループについて null を返します。

group by を持たない集合

定義により、スカラ集合はテーブル内のすべてのローに適用し、関数ごとにテーブル全体について単独の値を生成します。Transact-SQL 拡張機能はベクトル集合とともに拡張カラムを含めることを許可しますが、スカラ集合とともに拡張カラムを含めることも許可します。publishers テーブルの例を次に示します。

```
pub_id pub_name             city           state
------ -------------------  -------------- -----
0736   New Age Books        Boston         MA
0877   Binnet & Hardley     Washington     DC
1389   Algodata Infosystems Berkeley       CA
```

3 つのローがあります。次のクエリでは、テーブルの各ローに基づいて 3 つのローのスカラ集合を生成します。

```
select pub_id, count(pub_id)
from publishers

pub_id
---------- ---------
0736               3
0877               3
1389               3

(3 rows affected)
```

Adaptive Server は publishers を単独のグループとして扱い、スカラ集合は (単独グループ) テーブルに適用されます。結果は、スカラ集合に加えて select リストに含んだカラムのそれぞれについて、テーブルのすべてのローを表示します。

where 句は、スカラ集合に対して、ベクトル集合と同様に機能します。where 句は集合計算値に含まれるカラムを制限しますが、select リストで指定した各拡張カラムの結果に表示されるローには影響しません。次に例を示します。

```
select pub_id, count(pub_id)
from publishers
where pub_id < "1000"

pub_id
-------------- -----------
0736                   2
```

```
0877                    2
1389                    2

(3 rows affected)
```

group by に対する他の Transact-SQL 拡張機能と同様、この拡張機能は、特に大規模なテーブルについてのクエリや複数のテーブルのジョインを持つクエリの場合などに、理解が難しい結果を生成することがあります。

データのグループの選択：*having* 句

group by 句によって定義されたローを表示したり拒否したりするには、having 句を使用します。having 句は、where が select 句の条件を設定するのと同じ方法で、group by 句の条件を設定します。where 句では集合を含みませんが、having では含む点が異なります。次の例は有効です。

```
select title_id
from titles
where title_id like "PS%"
having avg(price) > $2.0
```

しかし次の例は、有効ではありません。

```
select title_id
from titles
where avg(price) > $20
--------------------
Msg 147, Level 15, State 1
Line 1:
An aggregate function may not appear in a WHERE clause unless
it is in a subquery that is in a HAVING clause, and the column
being aggregated is in a table named in a FROM clause outside
of the subquery.
```

having 句は select リスト内にある任意の項目を参照できます。

次の文は集合関数を持つ having 句の例です。titles テーブルにあるローを種類でグループ化しますが、本が 1 つしかないグループは除外します。

```
select type
from titles
group by type
having count (*) > 1

type
----------------
business
mod_cook
popular_comp
psychology
```

```
trad_cook

(5 rows affected)
```

次に集合を持たない having 句の例を示します。titles テーブルを種類でグループ化し、文字 “p” で始まる種類だけを返します。

```
select type
from titles
group by type
having type like "p%"

type
------------
popular_comp
psychology

(2 rows affected)
```

having 句に複数の条件を含める場合、それらの条件は and、or、または not で結合してください。たとえば、titles テーブルを出版社でグループ化し、本の平均価格が 18 ドル未満で ID 番号 (pub_id) が 0800 より大きく、合計で 15,000 ドルを超える前払い金を払った出版社だけを含めるには、次の文を使用します。

```
select pub_id, sum(advance), avg(price)
from titles
group by pub_id
having sum (advance) > $15000
  and avg(price) < 18
  and pub_id > "0800"

pub_id
------  ---------------- ----------------
0877          41,000.00            15.41

(1 row affected)
```

having、group by、where 句の相互関係

1 つのクエリに having、group by、where 句を含める場合、それぞれの句がローに影響する順序によって最終結果が決定されます。

- where 句はその探索条件を満たさないローを除外します。
- group by 句は残りのローを収集して、group by 式のユニークな値ごとに 1 つのグループにします。
- select リストで指定された集合関数は各グループの計算値を算出します。
- having 句は、その探索条件を満たさないローを最終結果から除外します。

次のクエリによって、集合関数を含む 1 つの select 文における where、group by、および having 句の機能がわかります。

```
select stor_id, title_id, sum(qty)
from salesdetail
where title_id like "PS%"
group by stor_id, title_id
having sum(qty) > 200

stor_id  title_id
-------  --------  -----------
5023     PS1372            375
5023     PS2091          1,845
5023     PS3333          3,437
5023     PS7777          2,206
6380     PS7777            500
7067     PS3333            345
7067     PS7777            250

(7 rows affected)
```

クエリは次の順序で実行されます。

1 where 句は、title_id が “PS” (心理学関連の本) で始まるローのみを識別しました。

2 group by が、共通 stor_id および title_id を使用してローを収集しました。

3 sum 集合が、各グループに対する本の総数を計算しました。

4 having 句が、最終結果の総計が 200 を超えないグループを除外しました。

このセクションの having 句の例はすべて SQL 規格に準じていたので、having 式中のカラムは単独の値を持つ必要があり、select リストまたは group by 句内にある必要がありました。しかし having に対する Transact-SQL 拡張機能では、select リストおよび group by 句にないカラムや式を使用できます。

次の例ではそれぞれのタイトルの種類について平均価格を判別しますが、sum 集合が結果に表示されなくても、総売り上げ高が 10,000 ドルを超えない種類を除外します。

```
select type, avg(price)
from titles
group by type
having sum(total_sales) > 10000

type
------------  ----------
business           13.73
mod_cook           11.49
popular_comp      21.48
trad_cook          15.96

(4 rows affected)
```

拡張機能は、カラムや式が select リストの一部ではあるが結果の一部ではないように動作します。having で集合を持たないカラムを指定し、そのカラムが select リストまたは group by 句の一部ではない場合、クエリはこの章で既に説明した「拡張」カラムの拡張機能と同様の結果を生成します。次に例を示します。

```
select type, avg(price)
from titles
group by type
having total_sales > 4000

type
------------  -----------
business            13.73
business            13.73
business            13.73
mod_cook            11.49
popular_comp        21.48
popular_comp        21.48
psychology          13.50
trad_cook           15.96
trad_cook           15.96

(9 rows affected)
```

拡張カラムと異なり、total_sales カラムは最終結果には表示されませんが、それぞれのタイプごとに表示されるローの数は、各タイトルの total_sales に依存します。クエリは、business の 3 タイトル、mod_cook の 1 タイトル、popular_comp の 2 タイトル、psychology の 1 タイトル、および trad_cook の 2 タイトルが、総売り上げで 4000 ドルを超えることを示します。

前述のように、Adaptive Server による拡張カラムの処理方法では、クエリが最終結果内で where 句を無視しているように見えます。where 条件が拡張カラムの結果に影響するようにするには、having 句内で条件を繰り返します。次に例を示します。

```
select type, advance, avg(price)
from titles
where advance > 5000
group by type
having advance > 5000

type           advance
-------------  ---------  --------
business       10,125.00      2.99
mod_cook       15,000.00      2.99
popular_comp    7,000.00     21.48
popular_comp    8,000.00     21.48
psychology      7,000.00
psychology      6,000.00     14.30
trad_cook       7,000.00     17.97
trad_cook       8,000.00     17.97

(8 rows affected)
```

group by を持たない *having* の使用

having 句を持つクエリは、group by 句も持つ必要があります。group by を省略すると、where 句で除外されないローはすべて単独のグループとして返されます。

where 句と having 句の間ではグループ化は行われないので、相互に独立して作用することはできません。having は複数のグループではなく単独のグループのローに影響するので where と同様に動作しますが、having 句は集合を使用できるという点が異なります

この例では、having 句を使用して、価格を平均化し、前払い金が 4,000 ドルを超えるタイトルを結果から除外し、価格が平均額を下回る結果を生成します。

```
select title_id, advance, price
from titles
where advance < 4000
having price > avg(price)

title_id        advance    price
-------------  ---------  --------
BU1032          5,000.00     19.99
BU7832          5,000.00     19.99
MC2222              0.00     19.99
PC1035          7,000.00     22.95
PC8888          8,000.00     20.00
PS1372          7,000.00     21.59
PS3333          2,000.00     19.99
TC3218          7,000.00     20.95

(8 rows affected)
```

また、select リストに集合を含むクエリから group by 句を省略できる Transact-SQL 拡張機能とともに having を使用することもできます。これらのスカラ集合関数は、テーブル内の複数のグループについてではなく、単独のグループとしてのテーブルについて値を計算します。

この例では group by 句を省略することによって、集合関数はテーブル全体について 1 つの値を計算します。having 句は結果グループから一致しないローを除外します。

```
select pub_id, count(pub_id)
from publishers
having pub_id < "1000"

pub_id
------ ----------------
0736                 3
0877                 3

(2 rows affected)
```

クエリ結果のソート：*order by* 句

order by 句を使用すると、最大 31 までのカラムによるクエリ結果のソートが可能になります。ソートはそれぞれ昇順 (asc) か降順 (desc) になります。どちらも指定されない場合は asc にデフォルトで設定されます。次のクエリは結果を pub_id でソートします。

```
select pub_id, type, title_id
from titles
order by pub_id
```

```
pub_id  type            title_id
------  ------------    --------
0736    business        BU2075
0736    psychology      PS2091
0736    psychology      PS2106
0736    psychology      PS3333
0736    psychology      PS7777
0877    UNDECIDED       MC3026
0877    mod_cook        MC2222
0877    mod_cook        MC3021
0877    psychology      PS1372
0877    trad_cook       TC3218
0877    trad_cook       TC4203
0877    trad_cook       TC7777
1389    business        BU1032
1389    business        BU1111
1389    business        BU7832
1389    popular_comp    PC1035
1389    popular_comp    PC8888
1389    popular_comp    PC9999

(18 rows affected)
```

複数カラム　order by 句に複数のカラムを指定すると、Adaptive Server はソートをネストします。次の文は stores テーブルのローをソートします。まず stor_id で降順に、次に payterms (昇順、desc が指定されていないため) でソートし、最後に country (昇順) でソートします。Adaptive Server はどのグループ内でも null 値を最初にソートします。

```
select stor_id, payterms, country
from stores
order by stor_id desc, payterms
```

```
stor_id payterms     country
------- ------------ ------------
8042    Net 30       USA
7896    Net 60       USA
7131    Net 60       USA
7067    Net 30       USA
7066    Net 30       USA
6380    Net 60       USA
```

```
5023    Net 60       USA

(7 rows affected)
```

カラムの位置番号　カラム名の代わりに、select リスト内のカラムの「位置番号」を使用できます。カラム名と select リスト番号が混在してもかまいません。次の文は、いずれも前述のものと同じ結果を生成します。

```
select pub_id, type, title_id
from titles
order by 1 desc, 2, 3
```

```
select pub_id, type, title_id
from titles
order by 1 desc, type, 3
```

SQL のほとんどのバージョンでは、order by 項目が select リスト内にある必要がありますが、Transact-SQL ではそのような制限はありません。title カラムは select リストにはありませんが、前述のクエリの結果は、このカラムで順序付けることができます。

注意　text、unitext、image カラムについては、order by は使用できません。

集合関数　集合関数は order by 句内で許可されていますが、どの order by カラムが union 式の影響を受けるかについてのあいまいさを避ける構文に従う必要があります。ただし、union 内のカラムの名前は、その union の最初 (左端) の部分から派生します。これは、union の最初の部分で指定されたカラム名だけを order by 句が使用することを意味します。

たとえば次の構文は、order by キーで識別されるカラムが明確に指定されているため、機能します。

```
select id, min(id) from tab
union
select id, max(id) from tab
ORDER BY 2
```

ただし、次の例では、エラー・メッセージが表示されます。

```
select id+2 from sysobjects
union
select id+1 from sysobjects
order by id+1
------------
Msg 104, Level 15, State1:
Line 3:
Order-by items must appear in the select list if the statement
contains set operators.
```

union の前後を入れ替えて文を並び替えると、正しく実行されます。

```
select id+1 from sysobjects
```

```
union
select id+2 from sysobjects
order by id+1
```

null 値　order by を使用すると、他のどの値よりも null 値が先になります。

大文字と小文字が混在するデータ　大文字と小文字が混在するデータでの order by 句の影響は、Adaptive Server にインストールされたソート順によって異なります。基本的な選択肢は、バイナリ・ソート順、辞書順、大文字と小文字を区別しないソート順です。sp_helpsort は、サーバのソート順を表示します。詳細については、『システム管理ガイド 第 1 巻』の「第 9 章 文字セット、ソート順、言語の設定」を参照してください。

制限事項　Adaptive Server では、order by リスト内のサブクエリまたは変数は許可されていません。

text、unitext、image カラムについては、order by は使用できません。

order by と group by

order by 句を使用して、group by の結果を特定の方法で順序付けることができます。

group by 句の後に、order by 句を置きます。たとえば、各種類の本の平均価格を調べ、その結果を平均価格で順序付けるには、次のような文になります。

```
select type, avg(price)
from titles
group by type
order by avg(price)

type
---------- ------------
UNDECIDED           NULL
mod_cook          11.49
psychology        13.50
business          13.73
trad_cook         15.96
popular_comp      21.48

(6 rows affected)
```

order by および *group by* の *select distinct* での使用

order by または group by を持つselect distinct クエリは、order by または group by カラムが select リスト内にない場合、重複する値を返します。次に例を示します。

```
select distinct pub_id
from titles
```

```
order by type

pub_id
------
0877
0736
1389
0877
1389
0736
0877
0877

(8 rows affected)
```

select リストにないカラムを含む order by または group by 句がクエリにある場合、Adaptive Server はこれらのカラムを処理中のカラムに隠しカラムとして追加します。order by または group by 句にリストされるカラムは、重複しないローのテストに含まれます。ANSI 規格に準拠するには、select リストに order by または group by カラムを含めます。次に例を示します。

```
select distinct pub_id, type
from titles
order by type

pub_id type
------ ------------
0877   UNDECIDED
0736   business
1389   business
0877   mod_cook
1389   popular_comp
0736   psychology
0877   psychology
0877   trad_cook

(8 rows affected)
```

グループ化したデータの計算：*compute* 句

compute 句は Transact-SQL 拡張機能です。これをロー集合関数とともに使用して、グループの計算値の小計を示すレポートを生成します。このようなレポートは通常、レポート生成プログラムによって生成されますが、計算値が compute 句で指定したグループ化 (“ブレーク”) で制御されてレポートに表示されるため、制御ブレーク・レポートと呼ばれます。

新規カラムとして表示される group by 句の集合結果とは異なり、これらの計算値はクエリ結果に追加のローとして表示されます。

compute 句を使用すると、ディテール・ローと計算ローを単一の select 文で参照できます。サブグループの計算値を計算したり、同一のグループの複数のロー集合関数を計算できます (「ロー集合関数と compute」(97 ページ) を参照してください)。

compute の一般的な構文は次のとおりです。

```
compute row_aggregate(column_name)
    [, row_aggregate(column_name)]...
    [by column_name [, column_name]...]
```

compute とともに使用できるロー集合関数は、sum、avg、min、max、count、count_big です。sum および avg は数値カラムでのみ使用できます。order by 句とは異なり、カラム名の代わりに select リストからのカラムの位置番号を使用することはできません。

注意　compute 句には、text、unitext、image カラムは使用できません。

クエリの compute 句に非常に多くの集合関数がある場合、システム・テストは失敗することがあります。各 compute 句に含めることのできる集合関数の個数の上限は 127 であり、compute 句に 127 個よりも多くの集合関数がある場合、クエリの実行時にエラー・メッセージが生成されます。

avg 集合関数は実際には sum 集合関数と count 集合関数の組み合わせでできているため、avg 集合関数 1 個は、上限の 127 個まで数えるときには 2 個の集合関数として数えられます。

次に 2 つのクエリとその結果を示します。最初のクエリは group by と集合を使用します。2 つめのクエリは compute とロー集合関数を使用します。結果の違いに注意してください。

```
select type, sum(price), sum(advance)
from titles
group by type

type
------------   -------    ----------
UNDECIDED         NULL         NULL
business         54.92    25,125.00
mod_cook         22.98    15,000.00
popular_comp     42.95    15,000.00
psychology       67.52    21,275.00
trad_cook        47.89    19,000.00

(6 rows affected)

select type, price, advance
from titles
order by type
compute sum(price), sum(advance) by type

type         price                     advance
------------ ------------------------ --------
```

```
UNDECIDED    NULL                      NULL

Compute Result:
---------------------- ------------------------
                  NULL                     NULL

type         price                advance
------------ -------------------- ----------
business     2.99                 10,125.00
business    11.95                  5,000.00
business    19.99                  5,000.00
business    19.99                  5,000.00

Compute Result:
---------------------- ------------------------
                54.92                25,125.00
type         price                  advance
------------ ---------------------- ---------
mod_cook      2.99                  15,000.00
mod_cook     19.99                       0.00

Compute Result:
---------------------- ------------------------
                22.98                15,000.00

type          price                advance
------------- -------------------- ------------
popular_comp  NULL                        NULL
popular_comp  20.00                   8,000.00
popular_comp  22.95                   7,000.00

Compute Result:
---------------------- ------------------------
                42.95                15,000.00

type         price                   advance
------------ ----------------------- --------
psychology    7.00                   6,000.00
psychology    7.99                   4,000.00
psychology   10.95                   2,275.00
psychology   19.99                   2,000.00
psychology   21.59                   7,000.00

Compute Result:
---------------------- ------------------------
                67.52                21,275.00

type         price                  advance
------------ ---------------------- --------
trad_cook    11.95                  4,000.00
trad_cook    14.99                  8,000.00
```

```
trad_cook    20.95                    7,000.00

Compute Result:
---------------------- -------------------------
                 47.89                19,000.00

(24 rows affected)
```

各計算値は 1 つのローとして扱われます。

ロー集合関数と *compute*

compute とともに使用するロー集合関数を、表 3-2 に示します。

表 3-2: compute 文で使用される集合

ロー集合関数	結果
sum	式中の値の合計
avg	式中の値の平均
max	式中の最も高い値
min	式中の最も低い値
count	integer として選択されたローの数
count_big	bigint として選択されたローの数

これらのロー集合関数は group by とともに使用できる集合関数と同じです。ただし、count(*) に相当するロー集合関数はありません。group by および count(*) によって生成されるまとめの情報を検出するには、by キーワードを指定せずに compute 句を使用します。

compute 句の規則

- Adaptive Server では distinct キーワードをロー集合関数とともに使用することはできません。
- compute 句内のカラムも select リストになければなりません。
- compute を含む文は通常のローを生成しないため、select into を compute 句と同じ文中で使用することはできません (「第 8 章 データベースおよびテーブルの作成」を参照)。
- compute を含む文は通常のローを生成しないという同じ理由で、insert 文内の select 文で compute 句を使用することはできません。
- by キーワードとともに compute を使用する場合は、order by 句も使用する必要があります。by の後にリストされるカラムは、order by の後にリストされるカラムと同じであるかまたはそのサブセットである必要があり、同じ式で始まる左から右への同じ順序で、どの式も省略してはなりません。

たとえば、次のような order by 句があるとします。

```
order by a, b, c
```

compute 句は次のいずれか、またはすべてになります。

```
compute row_aggregate (column_name) by a, b, c
compute row_aggregate (column_name) by a, b
compute row_aggregate (column_name) by a
```

compute 句は次のいずれかであってはなりません。

```
compute row_aggregate (column_name) by b, c
compute row_aggregate (column_name) by a, c
compute row_aggregate (column_name) by c
```

order by 句内のカラム名または式を使用してください。カラム見出しでソートすることはできません。

- 合計、合計カウントなどを生成するために、compute キーワードに by を付けないで使用できます。compute キーワードに by を付けない場合、order by 句はオプションになります。by を指定しない compute キーワードについては、「合計の生成：by を指定しない compute」(101 ページ) を参照してください。

compute に対する複数カラムの指定

by キーワードの後に複数のカラムをリストすると、1 つのグループがいくつかのサブグループに分割され、グループの各レベルにロー集合関数が適用されることによってクエリが影響を受けます。たとえば、各出版社から心理学関連の本の価格の合計を調べるクエリは、次のようになります。

```
select type, pub_id, price
from titles
where type = "psychology"
order by type, pub_id, price
compute sum(price) by type, pub_id

type         pub_id  price
------------ ------- -------------
psychology    0736           7.00
psychology    0736           7.99
psychology    0736          10.95
psychology    0736          19.99
Compute Result:
---------------
          45.93

type         pub_id  price
------------ ------- -------------
```

```
 psychology    0877          21.59

Compute Result:
---------------
          21.59

(7 rows affected)
```

複数の *compute* 句の使用

複数の compute 句を含めることによって、異なる集合を同じ文中で使用できます。たとえば、次のクエリは前述のクエリと似ていますが、出版社ごとに心理学関連の本の価格の合計を計算します。

```
select type, pub_id, price
from titles
where type = "psychology"
order by type, pub_id, price
compute sum(price) by type, pub_id
compute sum(price) by type

type        pub_id  price
----------- ------- --------------
psychology    0736           7.00
psychology    0736           7.99
psychology    0736          10.95
psychology    0736          19.99

Compute Result:
---------------
          45.93

type        pub_id  price
 ---------- ------- --------------
 psychology    0877         21.59

Compute Result:
---------------
          21.59

Compute Result:
---------------
           67.52

(8 rows affected)
```

複数のカラムへの集合の適用

1 つの compute 句は、同じ集合をいくつかのカラムに適用できます。次のクエリは料理関連の本の種類ごとに価格と前払い金の合計を検出します。

```
select type, price, advance
from titles
where type like "%cook"
order by type
compute sum(price), sum(advance) by type

type      price            advance
--------- ---------------- ---------------
mod_cook              2.99       15,000.00
mod_cook             19.99            0.00

Compute Result:
--------------- ---------------
          22.98       15,000.00

type      price            advance
--------- ---------------- ---------------
trad_cook            11.95        4,000.00
trad_cook            14.99        8,000.00
trad_cook            20.95        7,000.00

Compute Result:
--------------- ---------------
          47.89       19,000.00

(7 rows affected)
```

集合が適用されるカラムも select リストになければならないことに注意してください。

同じ *compute* 句内での異なる集合の使用

同じ compute 句内で異なる集合を使用できます。

```
select type, pub_id, price
from titles
where type like "%cook"
order by type, pub_id
compute sum(price), max(pub_id) by type

type        pub_id  price
----------- ------- --------------
mod_cook    0877            2.99
mod_cook    0877            19.99

Compute Result:
--------------- ----
          22.98 0877

type        pub_id  price
----------- ------- --------------
trad_cook   0877            11.95
trad_cook   0877            14.99
trad_cook   0877            20.95

Compute Result:
--------------- ----
          47.89 0877

(7 rows affected)
```

合計の生成：*by* を指定しない *compute*

by を指定せずに compute キーワードを使用して、総計、総カウントなどを生成できます。

次の文は 20 ドルを超えるすべての種類の本について価格および前払い金の総計を検出します。

```
select type, price, advance
from titles
where price > $20
compute sum(price), sum(advance)

type         price            advance
------------ ----------------  -------------
popular_comp            22.95      7,000.00
psychology              21.59      7,000.00
trad_cook               20.95        7,000.00

Compute Result:
--------------- ---------
```

```
          65.49 21,000.00

(4 rows affected)
```

by を指定した compute と by を指定しない compute を、同じクエリ内で使用できます。次に示すクエリは種類ごとに価格と前払い金の合計を検出してから、すべての種類の価格と前払い金の総計を計算します。

```
select type, price, advance
from titles
where type like "%cook"
order by type
compute sum(price), sum(advance) by type
compute sum(price), sum(advance)

type         price                advance
-----------  -----------------    ------------
mod_ cook                 2.99       15,000.00
mod_cook                 19.99            0.00

Compute Result:
--------------- ---------
          22.98 15,000.00

type         price                advance
-----------  -----------------    ------------
trad_cook                11.95        4,000.00
trad_cook                14.99        8,000.00
trad_cook                20.95        7,000.00

Compute Result:
--------------- ---------
          47.89 19,000.00

Compute Result:
--------------- ---------
          70.87 34,000.00

(8 rows affected)
```

クエリの結合：*union* 演算子

union 演算子は複数のクエリの結果を結合して 1 つの結果セットにします。Transact-SQL 拡張機能の union によって、次のタスクを実行できます。

- insert 文の select 句内で union を使用できます。
- select 文中に union があると、select 文の order by 句内に新規のカラム見出しを指定できます。

『リファレンス・マニュアル：コマンド』を参照してください。

図 3-1 は、2 つのテーブル、T1 および T2 を示しています。T1 には、2 つのカラム “a char(4)” および “b int” があります。T2 には、2 つのカラム “a char(4),” および “b int” があります。各テーブルには 3 つのローがあります。ロー 1 は、“a” カラムに “abc”、“b” カラムに “1” と表示します。T1 のロー 2 は、“a” カラムに “def”、“b” カラムに “2” と表示します。ロー 3 は、“a” カラムに “ghi”、“b (int)” カラムに “3” と表示します。テーブル T2 のロー 1 は、“a” カラムに “ghi”、“b” カラムに “3” と表示し、ロー 2 は、“a” カラムに “jkl”、“b” カラムに “4” と表示します。また、ロー 3 は、“a” カラムに “mno”、“b (int)” カラムに “5” と表示します。

図 3-1: クエリを結合する union

テーブル T1

a	b
char(4)	int
abc	1
def	2
ghi	3

テーブル T2

a	b
char(4)	int
ghi	3
jkl	4
mno	5

次に示すクエリは、2 つのテーブルの間で union を作成します。

```
create table T1 (a char(4), b int)
insert T1 values ("abc", 1)
insert T1 values ("def", 2)
insert T1 values ("ghi", 3)
create table T2 (a char(4), b int)
insert T2 values ("ghi", 3)
insert T2 values ("jkl", 4)
insert T2 values ("mno", 5)
select * from T1
union
select * from T2

a     b
----  ---------
abc           1
def           2
ghi           3
jkl           4
mno           5

(5 rows affected)
```

デフォルトでは、union 演算子は重複するローを結果セットから取り除きます。all オプションを使用してください。また、結果セット内のカラムが T1 にあるカラムと同じ名前であることにも注意してください。Transact-SQL 文中では union 演算子をいくつでも使用できます。次に例を示します。

```
x union y union z
```

デフォルトで、Adaptive Server では union 演算子を含む文を左から右に評価します。異なる評価順序を指定するには、カッコを使用します。

たとえば、次の 2 つの式は等価ではありません。

```
x union all (y union z)

(x union all y) union z
```

1 つめの式では、*y* と *z* の間の union での重複は削除されます。次に、そのセットと *x* の間の union では、重複は削除されません。2 つ目の式では、*x* と *y* の間の union では重複が含まれますが、それに続く *z* との union で削除されます。all はこの文の最終結果には影響しません。

union クエリのガイドライン

次に、union 文を使用するときのガイドラインを示します。

- union 文中の select リストは、すべて同じ数の式 (カラム名、算術式、集合関数など) を持つ必要があります。次の文は、1 つ目の select リストが 2 つ目のものより長いため、無効です。

```
create table stores_east
(stor_id char(4) not null,
stor_name varchar(40) null,
stor_address varchar(40) null,
city varchar(20) null,
state char(2) null,
country varchar(12) null,
postalcode char(10) null,
payterms varchar(12) null)
select stor_id, city, state from stores
union
select stor_id, city from stores_east
drop table stores_east
```

- すべてのテーブルの対応カラムや個々のクエリ内で使用されるカラムのサブセットは、データ型が同じであるか、2 つのデータ型間での暗黙的なデータ変換ができるか、または明示的なデータ変換が提供される必要があります。たとえば、char データ型のカラムと int データ型のカラムの間では、明示的な変換が提供されないかぎり、union は使用できません。ただし、money データ型のカラムと int データ型のカラムの間では、union を使用できます。union については、『リファレンス・マニュアル：コマンド』および『リファレンス・マニュアル：ビルディング・ブロック』の「第 1 章 システム・データ型とユーザ定義データ型」を参照してください。
- union はクエリ内で指定されている順に、カラムを 1 つずつ比較するため、union 文中のクエリ内の対応カラムは同じ順序で置かれる必要があります。たとえば、次のようなテーブルがあるとします。

図 3-2: カラムを比較する union

テーブル T3	
a int	b char(4)
1 2 3	abc def ghi

テーブル T4	
a char(4)	b int
abc def ghi	1 2 3

表 3-2は、2 つのテーブル、T3 および T4 を示しています。T3 には、2 つのカラム “a int” および “b char(4)” があります。T4 には、2 つのカラム “a char(4)” および “b int” があります。各テーブルには 3 つのローがあります。ロー 1 は、“a” カラムに “1”、“b” カラムに “abc” と表示します。ロー 2 は、“a” カラムに 2”、“b” カラムに “def” と表示します。ロー 3 は、“a” カラムに “3” および “b char” カラムに “ghi” と表示します。テーブル T4 のロー 1 は、“a” カラムに “abc”、“b” カラムに “1” と表示し、ロー 2 は、“a” カラムに “def”、“b” カラムに “2” と表示します。また、ロー 3 は、“a” カラムに “ghi”、“b(int)” カラムに “3” と表示します。

次のクエリを入力します。

```
select a, b from T3
union
select b, a from T4
```

クエリは、次の結果を生成します。

```
a           b
----------  ---
          1  abc
```

```
        2  def
        3  ghi

(3 rows affected)
```

しかし、次に示すクエリは、対応するカラムのデータ型に互換性がないため、エラー・メッセージを生成します。

```
select a, b from T3
union
select a, b from T4
drop table T3
drop table T4
```

float と int など、異なる (ただし互換性のある) データ型を union 文中で結合すると、Adaptive Server はこれらを最も精度の大きいデータ型に変換します。

- Adaptive Server は、union の結果として生成されたテーブルのカラム名を、union 文の 1 つ目のクエリから取得します。このため、結果セットに新規のカラム見出しを定義する場合は、1 つ目のクエリ内で行います。また、たとえば order by 文中など、結果セット内のカラムを新しい名前で参照する場合は、最初の select 文で、その方法によってカラムを参照します。

  正しいクエリを次に示します。

```
select Cities = city from stores
union
select city from authors
order by Cities
```

他の Transact-SQL コマンドでの *union* の使用

次に、union 文を他の Transact-SQL コマンドで使用するときのガイドラインを示します。

- union 文の最初のクエリには、最終的な結果セットを保持するテーブルを作成する into 句を含めることができます。たとえば次の文は、テーブル publishers、stores、および salesdetail の union を含む results というテーブルを作成します。

```
use mastersp_dboption pubs2, "select into", true
use pubs2
checkpoint
select pub_id, pub_name, city into results
from publishers
union
select stor_id, stor_name, city from stores
union
select stor_id, title_id, ord_num from salesdetail
```

into 句は 1 つ目のクエリ内でだけ使用できます。それ以外の場所に置くと、エラー・メッセージが返されます。

- order by 句と compute 句は、union 文の終わりでだけ使用して、最終結果の順序を定義したり、計算値を計算したりできます。これらの句を、union 文を構成する個々のクエリ内で使用することはできません。具体的には、insert...select ステートメント内では compute 句を使用できません。
- group by 句と having 句は、個々のクエリ内でだけ使用できます。最終結果セットに影響するようには使用できません。
- union 演算子を insert 文中で使用することもできます。次に例を示します。

```
create table tour (city varchar(20), state char(2))
insert into tour
    select city, state from stores
    union
    select city, state from authors
drop table tour
```

- Adaptive Server バージョン 12.5 からは、create view 文で union 演算子を使用できます。ただし、以前のバージョンの Adaptive Server を使用している場合は、create view 文で union 演算子を使用することはできません。
- text および image カラムについては、union 演算子は使用できません。
- union 演算子を含む文では、for browse 句を使用できません。

第 4 章

ジョイン：複数テーブルからのデータの検索

「ジョイン」演算とは、2 つ以上のテーブル (ビュー) に対して、それぞれの 1 つのカラムを指定して、ローごとにカラムの値を比較し、一致する値を持つローをリンクすることによって、その 2 つ以上のテーブル (ビュー) を比較することです。その後、新しいテーブルにその結果を表示します。ジョイン内に指定するテーブルは、1 つのデータベース内のものでも、別のデータベースにあるものでもかまいません。

多数のジョインをサブクエリとして指定できます。サブクエリも複数のテーブルを含みます。「第 5 章 サブクエリ：他のクエリ内でのクエリの使用」 を参照してください。

コンポーネント統合サービスが有効になっていると、リモート・サーバ間でジョインを行うことができます。『コンポーネント統合サービス・ユーザーズ・ガイド』を参照してください。

ジョインの動作

2 つ以上のテーブルをジョインするとき、比較するカラムは類似した値、つまり、同一のデータ型または類似したデータ型を使った値である必要があります。

ジョインには、等価ジョイン、ナチュラル・ジョイン、外部ジョインなど、いくつかのタイプがあります。等価ジョインは、最も一般的なジョインであり、等号を基にしています。次のジョインは同じ都市の作家と出版社の名前を検索します。

```
select au_fname, au_lname, pub_name
from authors, publishers
where authors.city = publishers.city
au_fname      au_lname      pub_name
--------      --------      --------------------
Cheryl        Carson        Algodata Infosystems
Abraham       Bennet        Algodata Infosystems

(2 rows affected)
```

このクエリは、publishers と authors という 2 つの別々のテーブルに含まれている情報を引き出すので、要求した情報を取り出すにはジョインが必要です。次の文は、city カラムをリンクとして使い、publishers テーブルと authors テーブルをジョインします。

```
where authors.city = publishers.city
```

ジョインの構文

ジョインは、select、update、insert、delete、またはサブクエリに埋め込むことができます。その他のジョインの制限と句はジョインの条件に従います。ジョインでは、次の構文を使用します。

select、*update*、*insert*、*delete*、またはサブクエリの開始
    from {*table_list* | *view_list*}
    where [not]
        [*table_name*.| *view_name*.]*column_name join_operator*
        [*table_name*.| *view_name*.]*column_name*
    [{and | or} [not]
        [*table_name*.|*view_name*.]*column_name join_operator*
         [*table_name*.|*view_name*.]*column_name*]...

select、*update*、*insert*、*delete*、またはサブクエリの終了

ジョインとリレーショナル・モデル

ジョイン演算は、データベース管理のリレーショナル・モデルにおける最も大きな特徴です。他のどのような機能にも増して、リレーショナル・データベース管理システムが他のデータベース管理システムと区別される機能は、ジョインです。

ネットワーク・システムや階層システムとしてよく知られている構造化データベース管理システムでは、データ値の間の関係はあらかじめ定義されています。データベースが設定されてしまうと、データ間の予期しない関係についてクエリを行うのは難しくなります。

リレーショナル・データベース管理システムでは、データ値の間の関係はデータベースの定義では記述されません。データ値の間の関係は、データが操作されたとき (データベースを作成したときではなく、データベースに「問い合わせた」とき) に明らかになります。このため、データベースを設定したときに予想していたことには関係なく、データベースに格納されたデータについて、思いつくあらゆる質問をすることができます。

「正規化規則」というデータベース設計の規則に従って、テーブルは、それぞれ、人、場所、事柄、または物といった一種のエンティティを記述します。そのために、2 種類以上のエンティティについての情報を比較しようとするときに、ジョイン演算が必要になるのです。異なるテーブルに格納されているデータ間の関係は、それらのデータをジョインすることによって発見されます。

この規則の当然の結果として、データベースに新しい種類のデータを追加するとき、ジョイン演算によって無制限の柔軟性が提供されます。各種のエンティティについてのデータを含む新しいテーブルは常に作成できます。新しいテーブルのフィールドが既存のテーブルにあるフィールドに似た値を持つ場合、ジョインによってその新しいテーブルを既存のテーブルにリンクできます。

ジョインの構造化

ジョイン文は、select 文のように、キーワード select で始まります。select キーワードの後に名前を指定したカラムは、クエリ結果に希望の順番で含まれるカラムです。この例では、authors テーブルからは作家の名前を含むカラムを、publishers テーブルからは出版社の名前を含むカラムを次のように指定します。

```
select au_fname, au_lname, pub_name
from authors, publishers
```

au_fname、au_lname、pub_name カラムは、属しているテーブルがはっきりとわかるので、修飾する必要はありません。しかし、ジョイン比較に使う city カラムは、authors テーブルと publishers テーブルの両方にその名前のカラムがあるので、次のように修飾する必要があります。

```
select au_fname, au_lname, pub_name
from authors, publishers
where authors.city = publishers.city
```

結果にはどちらの city カラムも出力されませんが、Adaptive Server は比較を実行するテーブルの名前を必要とします。

クエリに関係するテーブルのすべてのカラムを結果に入れるように指定するには、select にアスタリスク (*) を使います。たとえば、前述のジョイン・クエリで authors と publishers のすべてのカラムを含めるには、文は次のようになります。

```
select *
from authors, publishers
```

```
where authors.city = publishers.city
au_id       au_lname au_fname phone        address
city       state postalcode contract pub_id pub_name
city       state
----------- -------- -------- ------------ ---------------------
---------- ----- ---------- -------- ------ --------------------
---------- -----
238-95-7766 Carson   Cheryl   415 548-7723 589 Darwin Ln.
Berkeley   CA    94705      1        1389   Algodata Infosystems
Berkeley   CA
409-56-7008 Bennet   Abraham  415 658-9932 223 Bateman St
Berkeley   CA    94705      1        1389   Algodata Infosystems
Berkeley   CA

(2 rows affected)
```

出力は合計で 13 カラムずつの 2 つのローになります。ローが長いので、1 つのローが複数行になります。“*” が使われているときには、結果のカラムは常に、テーブルを作成した create 文に記述されている順序で表示されます。

select リストとジョインの結果には、ジョインされる両方のテーブルのカラムを含める必要はありません。たとえば、出版社の 1 つと同じ都市に住んでいる作家の名前を検索するときに、クエリに publishers テーブルのカラムを含める必要はありません。

```
select au_lname, au_fname
from authors, publishers
where authors.city = publishers.city
```

select 文の場合と同様、select リスト内のカラム名と from 句内のテーブル名はカンマで区切る必要があります。

from 句

ジョインする対象のテーブルとビューを指定するには、from 句を使います。これは、ジョインを行おうとしていることを Adaptive Server に示す句です。テーブルやビューは任意の順序でリストできます。テーブルの順序は、select リストの指定で select * を使ったときに表示される結果にだけ影響します。

クエリは最大で 50 個のテーブルと 46 個のワーク・テーブル (集合関数によって作成されたテーブルなど) を参照できます。50 個のテーブル制限には次のものが含まれます。

- from 句にリストされるテーブル (またはテーブルのビュー)
- 同じテーブルに対する複数の参照 (セルフジョイン) の各インスタンス
- サブクエリで参照されるテーブル
- into で作成されるテーブル
- from 句にリストされるビューによって参照されるベース・テーブル

次の例は、titles テーブルと publishers テーブルのカラムをジョインして、カリフォルニア州で出版されたすべての本の価格を 2 倍にしたものです。

```
begin tran
update titles
  set price = price * 2
  from titles, publishers
  where titles.pub_id = publishers.pub_id
  and publishers.state = "CA"
rollback tran
```

3 つ以上のテーブルやビューを含むジョインについては、「3 つ以上のテーブルのジョイン」(122 ページ)を参照してください。

テーブル名やビュー名は、所有者やデータベースの名前で修飾することができ、また、使いやすさを考えて相関名を指定することもできます。次に例を示します。

```
select au_lname, au_fname
from pubs2.blue.authors, pubs2.blue.publishers
where authors.city = publishers.city
```

ビューは、テーブルとまったく同じようにジョインすることができ、また、テーブルが使われる箇所ならどこでも使うことができます。ビューについては「第 12 章 ビュー：データへのアクセスの制限」を参照してください。この章の例では、テーブルだけを使用します。

where 句

where 句は、どのローを結果に入れるかを決定するために使います。where は、from 句で名前を指定したテーブルやビューの間の接続を指定します。カラムが属しているテーブルやビューがあいまいな場合には、カラム名を修飾します。次に例を示します。

```
where authors.city = publishers.city
```

この where 句によって、ジョインされるカラムの名前 (必要であればテーブル名で修飾します) と、ジョイン演算子 (多くの場合は等号、場合によっては「より大きい」または「より小さい」) を指定します。where 句構文の詳細については、「第 2 章 クエリ：テーブルからのデータの選択」を参照してください。

注意　ジョインで where 句を指定しないと、予期しない結果を得ることになります。where 句がないと、前述のジョイン・クエリは 2 個のローではなく、69 個のローを生成することになります。このような結果になる理由については、「ジョインの処理方法」(115 ページ)の項を参照してください。

ジョイン文の where 句に含めることができる条件は、異なるテーブルのカラムをリンクするための条件だけではありません。つまり、1 つの SQL 文の中にジョイン演算と select 演算を含めることができます。例については、「ジョインの処理方法」(115 ページ) を参照してください。

ジョイン演算子

等号を基にしてカラムを一致させるジョインは「等価ジョイン」と呼ばれます。「等価ジョイン」の詳細な定義については、等号を基にしていないジョインの例と、「等価ジョインとナチュラル・ジョイン」(116 ページ) の項を参照してください。

等価ジョインは次の比較演算子を使います。

表 4-1: ジョイン演算子

演算子	意味
=	等しい
>	より大きい
>=	以上
<	より小さい
<=	以下
!=	等しくない
!>	以下
!<	以上

関係演算子を使用するジョインは、ひとまとめにして「シータ・ジョイン」と呼ばれます。「外部ジョイン」に使われる他のジョイン演算子のセットについては、この章の後半で詳述します。外部ジョイン演算子は Transact-SQL の拡張機能であり、表 4-2 に示す働きをします。

表 4-2: 外部ジョイン演算子

演算子	アクション
*=	ジョインされたカラムが一致するローだけでなく、1 番目のテーブルにあるすべてのローを結果に含める。
=*	ジョインされたカラムが一致するローだけでなく、2 番目のテーブルにあるすべてのローを結果に含める。

ジョインされるカラムのデータ型

ジョインされるカラムは、すべて、同一または互換性のあるデータ型を持っている必要があります。データ型を暗黙的に変換できないカラムを比較するときは、convert 関数を使用します。ジョインされるカラムは、多くの場合、同じ名前ですが、同じでなくてもかまいません。

ジョインに使われるデータ型に互換性がある場合は、Adaptive Server が自動的にそれらのデータ型を変換します。たとえば、Adaptive Server はすべての数値型カラム (bigint、int、smallint、tinyint、unsigned bigint、unsigned int、unsigned smallint、decimal、float) の間、および、すべての文字型と日付のカラム (char、varchar、unichar、univarchar、nchar、nvarchar、datetime、date、time) の間の変換を行います。データ型変換の詳細については、「第 16 章 クエリでの Transact-SQL 関数の使用」および『リファレンス・マニュアル：ビルディング・ブロック』の「第 1 章 システム・データ型とユーザ定義データ型」を参照してください。

ジョインと、*text* カラムおよび *image* カラム

text または image の値を含むカラムにはジョインを使うことはできません。ただし、where 句によって、2 つのテーブルの text カラムの長さを比較することはできます。次に例を示します。

```
where datalength(textab_1.textcol) >
datalength(textab_2.textcol)
```

ジョインの処理方法

ジョインが処理される方法を知ることは、ジョインを理解することになり、また、ジョインを誤って指定して予期しない結果を得た場合に、その理由を見つける手だてにもなります。この項では、概念用語でジョインの処理を説明します。Adaptive Server が使用する実際の手順は、さらに洗練されたものです。

ジョインを処理する最初の手順は、テーブルの「直積」(それぞれのテーブルのローにおける可能なすべての組み合わせ) を形成することです。2 つのテーブルの直積におけるローの数は、1 番目のテーブルにあるローの数と 2 番目のテーブルにあるローの数を乗算したものです。

authors テーブルと publishers テーブルの直積は 69 (23 の作家と 3 の出版社による乗算) です。クエリで、select リストの複数のテーブルのカラムを指定し、from 句で複数のテーブルを指定し、where 句を指定しないと、直積を確認できます。たとえば、これまでの例で使用したジョインのいずれかで where 句を省略すると、Adaptive Server は 23 人の作家それぞれに 3 社の出版社それぞれを組み合わせて、69 のローすべてを返します。

```
select au_lname, au_fname
from authors, publishers
```

この直積には特に役に立つ情報は含まれていません。事実、これは誤解を生む原因ともいえます。直積では、データベース内の作家が全員、データベース内のすべての出版社と関係があるかのような誤解を招きます。

ジョインに where 句を含めることで、一致させるカラムとそれらを一致させる根拠を指定します。また、その他の制限を含めることもできます。Adaptive Server は、直積を形成した後、where 句にある条件を使うことでジョインを満たさないローを削除します。

たとえば、前述の例 (authors テーブルと publishers テーブルの直積) の where 句は、作家の住んでいる都市が出版社のある都市とは違うすべてのローを結果から取り除きます。

```
where authors.city = publishers.city
```

等価ジョインとナチュラル・ジョイン

等号 (=) を基にしたジョインは「等価ジョイン」と呼ばれます。等価ジョインは、ジョインされるカラムの値を等号で比較してから、ジョインされるテーブル内のすべてのカラムを結果に含めます。

次のクエリは等価ジョインの例です。

```
select *
from authors, publishers
where authors.city = publishers.city
```

この文の結果で、city カラムは 2 回出力されます。定義によると、等価ジョインの結果には 2 つの同一のカラムが含まれます。同じ情報を繰り返すことに意味はないので、これらのカラムのうちの 1 つを結果のクエリから取り除くことができます。このように繰り返されたカラムを取り除いた結果は「ナチュラル・ジョイン」と呼ばれます。

結果が city カラムでの publishers と authors のナチュラル・ジョインになるクエリは、次のようになります。

```
select publishers.pub_id, publishers.pub_name,
  publishers.state, authors.*
from publishers, authors
where publishers.city = authors.city
```

カラム publishers.city は結果には含まれません。

次に、ナチュラル・ジョインのもう 1 つの例を示します。

```
select au_fname, au_lname, pub_name
from authors, publishers
where authors.city = publishers.city
```

複数のジョイン演算子を使って、3 つ以上のテーブルをジョインしたり、3 つ以上のカラムの組みをジョインしたりできます。これらの「ジョイン式」は、通常 and で接続されますが、or も使用できます。

次に、and で接続されたジョインの例を 2 つ示します。1 番目の例は、本についての情報 (本の種類、作家、タイトル) を、本の種類別にリストします。複数の著者を持つ本の場合は、各作家に対して 1 つずつの複数のリストがあります。

```
select type, au_lname, au_fname, title
from authors, titles, titleauthor
where authors.au_id = titleauthor.au_id
and titles.title_id = titleauthor.title_id
order by type
```

2 番目の例は、同一の都市内および州内の作家と出版社の名前を検索します。

```
select au_fname, au_lname, pub_name
from authors, publishers
where authors.city = publishers.city
and authors.state = publishers.state
```

追加条件のあるジョイン

ジョイン・クエリの where 句には、ジョイン条件の他に選択基準を含めることもできます。たとえば、7500 ドルよりも多くの前払い金が支払われた本のすべてのタイトルと出版社名を検索するには、次のようにします。

```
select title, pub_name, advance
from titles, publishers
where titles.pub_id = publishers.pub_id
and advance > $7500

title                          pub_name              advance
-----------------------------  --------------------  ---------
You Can Combat Computer Stress!  New Age Books        10,125.00
The Gourmet Microwave          Binnet & Hardley      15,000.00
Secrets of Silicon Valley      Algodata Infosystems   8,000.00
Sushi, Anyone?                 Binnet & Hardley       8,000.00

(4 rows affected)
```

ジョインされるカラム (titles と publishers の pub_id) は select リストに指定する必要はありません。したがって、それらのカラムは結果に表示されていません。

ジョイン文には必要な数だけ選択基準を含めることができます。選択基準とジョイン条件の順序は、結果に影響を及ぼしません。

等号を基にしていないジョイン

2 つのカラムにある値をジョインするための条件は、等号以外でもかまいません。不等 (!=)、より大きい (>)、より小さい (<)、以上 (>=)、および以下 (<=) という他の任意の比較演算子を使うことができます。Transact-SQL では、演算子 !> と !< も使用できます。これらはそれぞれ <= および >= と等価です。

次に示すジョインの例は、New Age Books 社の州であるマサチューセッツ州よりもアルファベット順で後の州に住んでいる New Age 社の作家を検索します。

```
select pub_name, publishers.state,
   au_lname, au_fname, authors.state
from publishers, authors
where authors.state > publishers.state
and pub_name = "New Age Books"
```

```
pub_name         state   au_lname       au_fname         state
-------------    ------  -------------- -----------      -----
New Age Books    MA      Greene         Morningstar      TN
New Age Books    MA      Blotchet-Halls Reginald         OR
New Age Books    MA      del Castillo   Innes            MI
New Age Books    MA      Panteley       Sylvia           MD
New Age Books    MA      Ringer         Anne             UT
New Age Books    MA      Ringer         Albert           UT

(6 rows affected)
```

次の例は、>= ジョインと < ジョインを使って、本の合計販売数を基に roysched テーブルの正しい royalty を検索します。

```
select t.title_id, t.total_sales, r.royalty
from titles t, roysched r
where t.title_id = r.title_id
and t.total_sales >= r.lorange
and t.total_sales < r.hirange
```

```
title_id       total_sales      royalty
--------       -----------      -------
BU1032               4095           10
BU1111               3876           10
BU2075               1872           24
BU7832               4095           10
MC2222               2032           12
MC3021              22246           24
PC1035               8780           16
PC8888               4095           10
PS1372                375           10
PS2091               2045           12
PS2106                111           10
PS3333               4072           10
PS7777               3336           10
TC3218                375           10
TC4203              15096           14
```

```
TC7777                4095           10

(16 rows affected)
```

セルフジョインと相関名

1 つのテーブルの中の同じカラム内の値を比較するジョインは「セルフジョイン」と呼ばれます。1 つのテーブルが 2 つの役割で表示されますが、この 2 つの役割を区別するために、エイリアスまたは相関名を使います。

たとえば、カリフォルニア州オークランドで、同じ郵便番号の地域に住む作家を検索するためにセルフジョインを使うことができます。このクエリは、authors テーブルとそれ自体とのジョインを含むので、authors テーブルは 2 つの役割を持ちます。これらの役割を区別するために、from 句で、au1 と au2 のように、2 つの異なる相関名を authors テーブルに一時的かつ任意に指定できます。これらの相関名は、そのクエリ内の他の場所でもカラム名の修飾に使用されます。セルフジョイン文は次のようになります。

```
select au1.au_fname, au1.au_lname,
au2.au_fname, au2.au_lname
from authors au1, authors au2
where au1.city = "Oakland" and au2.city = "Oakland"
and au1.state = "CA" and au2.state = "CA"
and au1.postalcode = au2.postalcode
au_fname     au_lname     au_fname    au_lname
---------    ----------- --------    --------
Marjorie     Green       Marjorie    Green
Dick         Straight    Dick        Straight
Dick         Straight    Dirk        Stringer
Dick         Straight    Livia       Karsen
Dirk         Stringer    Dick        Straight
Dirk         Stringer    Dirk        Stringer
Dirk         Stringer    Livia       Karsen
Stearns      MacFeather  Stearns     MacFeather
Livia        Karsen      Dick        Straight
Livia        Karsen      Dirk        Stringer
Livia        Karsen      Livia       Karsen

(11 rows affected)
```

from 句でエイリアスをリストするときは、この例のように select リストで指定した順に行うようにしてください。クエリによっては、異なる順序でエイリアスをリストすると、結果があいまいになる可能性があります。

結果の中で作家が一致するローを削除し、作家の順序が逆になっているだけで内容は重複しているローを削除するために、セルフジョイン・クエリに次のような追加を行うことができます。

```
select au1.au_fname, au1.au_lname,
```

```
au2.au_fname, au2.au_lname
from authors au1, authors au2
where au1.city = "Oakland" and au2.city = "Oakland"
and au1.state = "CA" and au2.state = "CA"
and au1.postalcode = au2.postalcode
and au1.au_id < au2.au_id

au_fname    au_lname     au_fname    au_lname
---------   ----------   ---------   ---------
Dick        Straight     Dirk        Stringer
Dick        Straight     Livia       Karsen
Dirk        Stringer     Livia       Karsen

(3 rows affected)
```

これで、Dick Straight、Dirk Stringer、Livia Karsen が同じ郵便番号を持っていることがはっきりしました。

不等価ジョイン

不等価ジョインは、セルフジョインで返されるローに制限を加えるときに便利です。次の例では、不等価ジョインとセルフジョインはさまざまな価格の安い本 (15 ドルよりも安い本) が 2 冊以上あるカテゴリを検索します。

```
select distinct t1.type, t1.price
from titles t1, titles t2
where t1.price < $15
and t2.price < $15
and t1.type = t2.type
and t1.price != t2.price
type        price
----------  -----
business     2.99
business    11.95
psychology   7.00
psychology   7.99
psychology  10.95
trad_cook   11.95
trad_cook   14.99

(7 rows affected)
```

式 “not *column_name* = *column_name*” は “*column_name* != *column_name*” と等価です。

次の例は、セルフジョインと組み合わせた不等価ジョインを使っています。ここでは、同じ title_id を持つが au_id 番号が異なるローが 2 つ以上ある titleauthor テーブルのすべてのロー、つまり、作家が複数いる本を検索します。

```
select distinct t1.au_id, t1.title_id
```

```
from titleauthor t1, titleauthor t2
where t1.title_id = t2.title_id
and t1.au_id != t2.au_id
order by t1.title_id
au_id            title_id
-----------      --------
213-46-8915      BU1032
409-56-7008      BU1032
267-41-2394      BU1111
724-80-9391      BU1111
722-51-5454      MC3021
899-46-2035      MC3021
427-17-2319      PC8888
846-92-7186      PC8888
724-80-9391      PS1372
756-30-7391      PS1372
899-46-2035      PS2091
998-72-3567      PS2091
267-41-2394      TC7777
472-27-2349      TC7777
672-71-3249      TC7777

(15 rows affected)
```

titles のそれぞれの本に対して、次の例は同じ種類で価格の違う他の本をすべて検索します。

```
select t1.type, t1.title_id, t1.price, t2.title_id, t2.price
from titles t1, titles t2
where t1.type = t2.type
and t1.price != t2.price
```

不等価ジョインとサブクエリ

不等価ジョインのクエリによる限定が十分ではないため、サブクエリで置き換える必要があることもあります。たとえば、出版社のない都市に住んでいる作家の名前をリストするとします。わかりやすくするために、姓が “A”、“B”、“C” のいずれかで始まる作家にこのクエリを限定します。不等価ジョインは次のようになります。

```
select distinct au_lname, authors.city
from publishers, authors
where au_lname like "[ABC]%"
and publishers.city != authors.city
```

しかし結果は次のようになり、これは前述の質問の答えではありません。

```
au_lname             city
----------------     ------------
Bennet               Berkeley
Carson               Berkeley
```

```
Blotchet-Halls       Corvallis

(3 rows affected)
```

システムでは、このクエリを「出版社がない都市に住んでいる作家の名前を検索せよ」という意味であると解釈します。Berkeley または Corvalis に住んでいる作家だけが、出版社がない都市という条件に適合します。

この場合、システムがジョインを処理するとき、最初に適格な組み合わせすべてを検索してから他の条件を評価するために、このクエリは希望しない結果を返します。必要な結果を取得するには、サブクエリを使用します。サブクエリはまず最初に不適格なローを削除してから残りの制限を実行します。

次に示すのが正しい文になります。

```
select distinct au_lname, city
from authors
where au_lname like "[ABC]%"
and city not in
(select city from publishers
where authors.city = publishers.city)
```

これで結果は希望するものになります。

```
au_lname             city
-------------        ------------
Blotchet-Halls       Corvallis

(1 row affected)
```

サブクエリの詳細については、「第 5 章 サブクエリ：他のクエリ内でのクエリの使用」を参照してください。

3 つ以上のテーブルのジョイン

3 つ以上のテーブルをジョインすることが役立つ場合があります。その良い例として、pubs2 の titleauthor テーブルの例があります。特定の種類のすべての本のタイトルと、その作家の名前を検索するには、次のクエリを使用します。

```
select au_lname, au_fname, title
from authors, titles, titleauthor
where authors.au_id = titleauthor.au_id
and titles.title_id = titleauthor.title_id
and titles.type = "trad_cook"
au_lname       au_fname     title
-------------- -----------  ------------------------
Panteley       Sylvia       Onions, Leeks, and Garlic: Cooking
                             Secrets of the Mediterranean
Blotchet-Halls Reginald     Fifty Years in Buckingham Palace
                            Kitchens
```

```
O'Leary         Michael       Sushi, Anyone?
Gringlesby      Burt          Sushi, Anyone?
Yokomoto        Akiko         Sushi, Anyone?

      (5 rows affected)
```

from 句にあるテーブルの 1 つ、titleauthor には、結果に表示されるカラムが 1 つもありません。さらに、ジョインされるカラム (au_id と title_id) はどれも結果には表示されません。それでもなお、このジョインは titleauthor を中間のテーブルとして使う場合にのみ可能になります。

また、同じ文の中に 3 つ以上のカラムの組をジョインすることができます。たとえば、次のクエリは title_id、そのタイトルの合計販売数とその範囲、および、結果の印税を示しています。

```
select titles.title_id, total_sales, lorange, hirange, royalty
from titles, roysched
where titles.title_id = roysched.title_id
and total_sales >= lorange
and total_sales < hirange

title_id    total_sales   lorange hirange  royalty
--------    -----------   ------- -------  -------
 BU1032            4095         0    5000       10
 BU1111            3876         0    4000       10
 BU2075           18722     14001   50000       24
 BU7832            4095         0    5000       10
 MC2222            2032      2001    4000       12
 MC3021            2224     12001   50000       24
 PC1035            8780      4001   10000       16
 PC8888            4095         0    5000       10
 PS1372             375         0   10000       10
 PS2091            2045      1001    5000       12
 PS2106             111         0    2000       10
 PS3333            4072         0    5000       10
 PS7777            3336         0    5000       10
 TC3218             375         0    2000       10
 TC4203           15096      8001   16000       14
 TC7777            4095         0    5000       10

(16 rows affected)
```

1 つの文の中で複数のジョイン演算子があるとき、3 つ以上のテーブルをジョインするか、3 つ以上のカラムの組をジョインする場合には、「ジョイン式」は前述の例で示すようにほとんど常に and で接続されます。ただし、or で接続することも有効です。

外部ジョイン

一致するローがあるかどうかに関係なく、すべてのローを含むジョインは「外部ジョイン」と呼ばれます。Adaptive Server は、左右両方の外部ジョインをサポートします。たとえば、次のクエリは titles テーブルと titleauthor テーブルを title_id カラムでジョインしています。

```
select *
from titles, titleauthor
where titles.title_id *= titleauthor.title_id
```

Sybase は Transact-SQL と ANSI の両方の外部ジョインをサポートしています。Transact-SQL の外部ジョインは *= コマンドを使って左外部ジョインを示し、=* コマンドを使って右外部ジョインを示します。Transact-SQL の外部ジョインは、Transact-SQL 言語の一部として Sybase が作成したものです。「Transact-SQL 外部ジョイン」(140 ページ) を参照してください。

ANSI 外部ジョインは left join と right join のキーワードを使って、それぞれ左ジョインと右ジョインを示しています。Sybase は ANSI 規格と完全に互換性を持つ外部ジョイン構文を実装しています。「ANSI の内部ジョインと外部ジョイン」(126 ページ) を参照してください。次に示すのは、前述の例を ANSI 外部ジョインに書き直したものです。

```
select *
from titles left join titleauthor
on titles.title_id = titleauthor.title_id
```

内部テーブルと外部テーブル

「外部テーブル」と「内部テーブル」という用語は、外部ジョインにおけるテーブルの配置を述べています。

- 「左ジョイン」では、「外部テーブル」と「内部テーブル」は、それぞれ左と右のテーブルのことです。また、外部テーブルと内部テーブルを、それぞれ、ロー保持テーブルと null 供給テーブルともいいます。
- 「右ジョイン」では、外部テーブルは右のテーブル、内部テーブルは左のテーブルです。

たとえば、次のクエリでは、T1 は外部テーブルで T2 は内部テーブルです。

```
T1 left join T2
T2 right join T1
```

また、Transact-SQL 構文を使うと次のようになります。

```
T1 *= T2
T2 =* T1
```

外部ジョインの制限事項

テーブルが外部ジョインの内部メンバである場合、テーブルを外部ジョイン句と通常のジョイン句の両方に置くことはできません。次のクエリは、salesdetail テーブルが外部ジョインと通常のジョイン句の両方の一部なので、失敗します。

```
select distinct sales.stor_id, stor_name, title
from sales, stores, titles, salesdetail
where qty > 500
and salesdetail.title_id =* titles.title_id
and sales.stor_id = salesdetail.stor_id
and sales.stor_id = stores.stor_id

Msg 303, Level 16, State 1:
Server 'FUSSY', Line 1:
The table 'salesdetail' is an inner member of an outer-join
clause.This is not allowed if the table also participates in a
regular join clause.
```

書籍の販売数が 500 部よりも多い書店の名前を知りたい場合は、2 番目のクエリを使う必要があります。外部ジョインと外部ジョインの内部テーブルからのカラムに対する修飾を持つクエリを実行した場合、その結果が予想した結果とは異なることもあります。クエリ内の修飾は返されるローの数は制限しませんが、null 値を含むローに影響を与えます。修飾に一致しないローに対して、それらのローの内部テーブル内のカラムに null 値が表示されます。

外部ジョインで使用されるビュー

外部ジョインでビューを定義してから、外部ジョインの内部テーブルからのカラムに対する修飾を使用してそのビューに問い合わせた場合、その結果が予想していたものとは異なることがあります。クエリは内部テーブルのすべてのローを返します。修飾を満たさないローは、それらのローの適切なカラムに null 値を示します。

次の規則は、ジョイン・ビューからカラムに対して実行できる更新の種類を決定します。

- delete 文はジョイン・ビューでは許可されません。
- insert 文は with check option で作成されたジョイン・ビューには許可されません。
- update 文は with check option のジョイン・ビューで使用できます。影響を受けるカラムが、複数のテーブルからのカラムを含む式の where 句に含まれている場合、更新は失敗します。
- ジョイン・ビューからローを挿入または更新した場合、影響を受けるすべてのカラムは同一のベース・テーブルに属する必要があります。

ANSI の内部ジョインと外部ジョイン

テーブルをジョインするための ANSI 構文を次に示します。

```
left_table [inner | left [outer] | right [outer]] join right_table
on left_column_name  = right_column_name
```

左のテーブルと右のテーブルの間のジョインの結果は「ジョイン・テーブル」と呼ばれます。ジョイン・テーブルは from 句で定義されます。次に例を示します。

```
select titles.title_id, title, ord_num, qty
from titles left join salesdetail
on titles.title_id = salesdetail.title_id
title_id title               ord_num                qty
---------------------------- ---------------------- -----
BU1032  The Busy Executive   AX-532-FED-452-2Z7   200
BU1032  The Busy Executive   NF-123-ADS-642-9G3   1000
. . .
TC7777  Sushi, Anyone?       ZD-123-DFG-752-9G8   1500
TC7777  Sushi, Anyone?       XS-135-DER-432-8J2   1090
(118 rows affected)
```

ANSI のジョイン構文では、次のいずれも表現できます。

- 「内部ジョイン」。内部ジョインでは、ジョイン・テーブルは on 句の条件を満たす内部テーブルと外部テーブルのローだけを含みます。「ANSI 内部ジョイン」(128 ページ) を参照してください。内部ジョインを含むクエリの結果セットには、on 句の条件を満たさない外部テーブルのローに関して、null が供給されたローは含まれません。
- 「外部ジョイン」。外部ジョインでは、ジョイン・テーブルは on 句の条件を満たすかどうかに関係なく、外部テーブルのすべてのローを含みます。ローが on 句の条件を満たさない場合、内部テーブルからの値が null 値としてジョイン・テーブルに格納されます。ANSI 外部ジョインの where 句がクエリ結果に含まれるローを限定します。「ANSI 外部ジョイン」(131 ページ) を参照してください。

注意 ANSI 構文ではビューをジョインすることもできます。

Sybase ANSI のジョイン構文は using 句をサポートしません。

ANSI ジョインにおける相関名とカラム参照規則

次に示すのは、ANSI ジョインでの相関名とカラム参照規則です。「セルフジョインと相関名」(119 ページ) を参照してください。

- テーブルまたはビューがカラムまたはビューに対して相対名参照を使用する場合、そのカラムやビューに対する参照は、テーブル名やビュー名ではなく同じ相関名を常に使用する必要があります。つまり、クエリ内のテーブルに相関名で名前を付けて、そのテーブル名を後で使用することはできません。次の例は、相関名 t を正しく使って、pub_id カラムが指定されているテーブルを指定しています。

  ```
  select title, t.pub_id, pub_name
  from titles t left join publishers p
  on t.pub_id = p.pub_id
  ```

  しかし、次の例はクエリの on 句で titles テーブル用の相関名 (t.pub_id) ではなく、誤ってテーブル名を使用しているので、エラー・メッセージが表示されます。

  ```
  select title, t.pub_id, pub_name
  from titles t left join publishers p
  on titles.pub_id = p.pub_id
  Msg 107, Level 15, State 1:
  Server 'server_name', Line 1:
  The column prefix 't' does not match with a table name or
  alias name used in the query.Either the table is not
  specified in the FROM clause or it has a correlation name
  which must be used instead.
  ```

- on 句に指定された制限は次の項目を参照できます。
  - ジョイン・テーブルの参照で指定されたカラム
  - ANSI ジョインに含まれるジョイン・テーブル (ネスト・ジョイン内など) に指定されたカラム
  - 外部クエリ・ブロックで指定されたテーブルに対するサブクエリ内の相関名
- on 句で指定された条件は、別の ANSI ジョインを含む ANSI ジョイン内に導入されたカラムを参照できません (通常、2 番目のジョインによって生成されたジョイン・テーブルが 1 番目のジョインとジョインされるとき)。

  次に示すのは、エラーとなる不正なカラム参照の例です。

```
select *
from titles left join titleauthor
on titles.title_id=roysched.title_id    /*join #1*/
left join roysched
on titleauthor.title_id=roysched.title_id    /*join #2*/
where titles.title_id != "PS7777"
```

roysched.title_id カラムは 2 番目のジョインまで導入されないので、1 番目の左ジョインはこのカラムを参照できません。このクエリは、次のように正しく書き直すことができます。

```
select *
from titles
left join (titleauthor
left join roysched
on titleauthor.title_id = roysched.title_id) /*join #1*/
on titles.title_id = roysched.title_id      /*join #2*/
where titles.title_id != "PS7777"
```

また、次のように記述することもできます。

```
select title, price, titleauthor.au_id, titleauthor.title_id, pub_name,
publishers.city
from roysched, titles
left join titleauthor
on roysched.title_id=titleauthor.title_id
left join authors
on titleauthor.au_id=roysched.au_id, publishers
```

このクエリでは、roysched テーブルも publishers テーブルも左ジョインの一部ではありません。このため、どちらの左ジョインも、その on 句条件の一部として roysched テーブルと publishers テーブルのカラムを参照できません。

ANSI 内部ジョイン

制限を満たすジョイン・テーブルのローだけを含む結果セットを生成するジョインは「内部ジョイン」と呼ばれます。ジョイン制限を満たさないローは、ジョイン・テーブルには含められません。ジョイン・テーブルに対してテーブルの 1 つのすべてのローを含めることが必要な場合は、制限を満たすかどうかに関係なく、外部ジョインを使用してください。「ANSI 外部ジョイン」(131 ページ) を参照してください。

Adaptive Server は Transact-SQL の内部ジョインと ANSI の内部ジョインの両方の使用をサポートします。Transact-SQL の内部ジョインを使用するクエリは、ジョインされるテーブルをカンマで分け、where 句内でジョインの比較と制限をリストします。次に例を示します。

```
select au_id, titles.title_id, title, price
from titleauthor, titles
where titleauthor.title_id = titles.title_id
and price > $15
```

「ジョインの構造化」(111 ページ) を参照してください。

ANSI 規格の inner join 構文は次のようになります。

```
select select_list
    from table1 inner join table2
     on join_condition
```

たとえば、次の inner join の用法は前述の Transact-SQL ジョインと同じです。

```
select au_id, titles.title_id, title, price
from titleauthor inner join titles
on titleauthor.title_id = titles.title_id
and price > 15
au_id         title_id  title                        price
----------    --------  -------------------------    -----
213-46-8915   BU1032    The Busy Executive's Datab  19.99
409-56-7008   BU1032    The Busy Executive's Datab  19.99
. . .
172-32-1176   PS3333    Prolonged Data Deprivation  19.99
807-91-6654   TC3218    Onions, Leeks, and Garlic:  20.95
(11 rows affected)
```

ジョインを記述する 2 つの方法である ANSI と Transact-SQL は同じものです。たとえば、次に示す 2 つのクエリで生成される結果セットは同じになります。

```
select title_id, pub_name
from titles, publishers
where titles.pub_id = publishers.pub_id
```

および

```
select title_id, pub_name
from titles left join publishers
on titles.pub_id = publishers.pub_id
```

内部ジョインは update 文または delete 文の一部にできます。たとえば、次のクエリはカリフォルニア州で出版されるすべての本の価格を 1.25 倍にします。

```
begin tran
update titles
  set price = price * 1.25
  from titles inner join publishers
  on titles.pub_id = publishers.pub_id
  and publishers.state = "CA"
```

内部ジョインのジョイン・テーブル

ANSI ジョインは、クエリ内でどのテーブルまたはビューをジョインするのかを指定します。ANSI ジョインで指定されたテーブル参照がジョイン・テーブルを構成します。たとえば、次のクエリのジョイン・テーブルは title、price、advance、royaltyper カラムを含んでいます。

```
select title, price, advance, royaltyper
from titles inner join titleauthor
on titles.title_id = titleauthor.title_id
title          price     advance      royaltyper
-----------    -------   ----------   ----------
The Busy...    19.99     5,000.00     40
The Busy...    19.99     5,000.00     60
. . .
```

```
Sushi, A...     14.99     8,000.00     30
Sushi, A...     14.99     8,000.00     40
(25 rows affected)
```

ジョインされたテーブルが ANSI 内部ジョインのテーブル参照として使用される場合、これは「ネストされた」内部ジョインになります。ANSI のネストされた内部ジョインは ANSI 外部ジョインと同じ規則に従います。

クエリは、次のものを含めて、union の両側にそれぞれ最大で 50 個のユーザ・テーブル (または 14 個のワーク・テーブル) を参照できます。

- from 句にリストされるベース・テーブルまたはビュー
- 同一のテーブルに対するそれぞれの相関参照 (セルフジョイン)
- サブクエリで参照されるテーブル
- ビューまたはネストされたビューで参照されるベース・テーブル
- into で作成されるテーブル

ANSI 内部ジョインの *on* 句

ANSI 内部ジョインの on 句は、テーブルやビューがジョインされるときに使われる条件を指定します。テーブル内にあるどのカラムでもジョインを使用できますが、ジョインされるカラムにインデックスが設定されている方がパフォーマンスは良くなります。多くの場合、カラムやそのカラムが属しているテーブルをユニークに識別するには修飾子 (テーブルや相関名) を使う必要があります。次に例を示します。

```
from titles t left join titleauthor ta
on t.title_id = ta.title_id
```

この on 句は、両方のテーブルから一致しない title_id ローを取り除きます。「セルフジョインと相関名」(119 ページ) を参照してください。

on 句は、次のクエリの 3 行目と 4 行目のように、多くの場合 ANSI ジョイン・テーブルを比較します。

```
select title, price, pub_name
from titles inner join publishers
on titles.pub_id = publishers.pub_id
and total_sales > 300
```

この on 句に指定されているジョイン制限は、販売数が 300 を超えないすべてのローをジョイン・テーブルから削除します。また、on 句には、and 修飾子を含めて、クエリの 4 行目にあるように探索引数をさらに指定できます。

ANSI 内部ジョインは、条件が on 句に配置されても where 句に配置されても (外部ジョインでネストされない限り)、結果セットを同様に制限します。そのため、次の 2 つのクエリは同じ結果セットを生成します。

```
select stor_name, stor_address, ord_num, qty
from salesdetail inner join stores
```

```
on salesdetail.stor_id = stores.stor_id
where qty > 3000
```

および

```
select stor_name, stor_address, ord_num, qty
from salesdetail inner join stores
on salesdetail.stor_id = stores.stor_id
and qty > 3000
```

通常、クエリは where 句に制限が指定される方が読みやすくなります。これは、ジョイン・テーブルのどのローが結果セットに含まれるのかを明確にユーザに示すためです。

ANSI 外部ジョイン

外部テーブルのすべてのローを含むジョイン・テーブルを作成するジョインは、on 句が一致するローを生成するかどうかに関係なく、「外部ジョイン」と呼ばれます。内部ジョインと等価ジョインは、ジョイン句に一致する値があるテーブルにあるローだけを含む結果セットを生成します。しかし、一致するローだけでなく、2 番目のテーブルにある一致しないローがあるテーブルのローも含めたい場合があります。このようなタイプのジョインが外部ジョインです。外部ジョインでは on 句条件に一致しないローは、外部ジョインの内部テーブル用に null が供給された値で、ジョインされるテーブルに含まれます。内部テーブルを null 供給メンバともいいます。

on 句と where 句のどちらが述語を含むかをはっきりと指定できるため、使用するアプリケーションで ANSI 外部ジョインを使用することをおすすめします。

この項では ANSI 外部ジョインだけを説明します。Transact-SQL 外部ジョインについては「Transact-SQL 外部ジョイン」(140 ページ) を参照してください。

注意　ANSI 外部ジョインを含むクエリには Transact-SQL 外部ジョインを入れることができず、逆に、Transact-SQL 外部ジョインを含むクエリには ANSI 外部ジョインを入れることはできません。ただし、ANSI 外部ジョインのクエリは Transact-SQL 外部ジョインを含むビューを参照することはでき、また、逆に Transact-SQL 外部ジョインのクエリは ANSI 外部ジョインを含むビューを参照することはできます。

ANSI 外部ジョインの構文は次のとおりです。

```
select select_list
    from table1 {left | right} [outer] join table2
    on predicate
    [join restriction]
```

左ジョインにはジョイン句の左側にリストされているテーブル参照のローがすべて保持され、右ジョインにはジョイン句の右側にあるテーブル参照のローがすべて保持されます。左ジョインでは、左テーブル参照を外部テーブルまたはロー保持テーブルと呼びます。

次の例は、自分の出版社と同じ都市に住んでいる作家を判断します。

```
select au_fname, au_lname, pub_name
from authors left join publishers
on authors.city = publishers.city
au_fname   au_lname    pub_name
---------  ----------  -------------
Johnson    White       NULL
Marjorie   Green       NULL
Cheryl     Carson      Algodata Infosystems
. . .
Abraham    Bennet      Algodata Infosystems
. . .
Anne       Ringer      NULL
Albert     Ringer      NULL
(23 rows affected)
```

結果セットには authors テーブルのすべての作家が含まれます。自分の出版社と同じ都市には住んでいない作家は pub_name カラムに null 値を生成します。出版元と同じ都市に住んでいる著者、Cheryl Carson および Abraham Bennet についてのみ、pub_name カラムに null 以外の値が生成されます。

from 句でテーブルの配置を逆にすることによって、左外部ジョインを右外部ジョインに書き換えることができます。また、select 文で “select *” を指定した場合にはすべてのカラム名の明示的なリストを作成する必要があります。このリストを作成しないと、結果セット内のカラムは書き換えられたクエリとは違う順序になります。

次に示しているのは、前述の例を右外部ジョインとして書き換えたものであり、前述の左外部ジョインと同じ結果セットを生成します。

```
select au_fname, au_lname, pub_name
from publishers right join authors
on authors.city = publishers.city
```

述語は *on* 句に入れるべきか、*where* 句に入れるべきか

ANSI 外部ジョインの結果セットは、その制限を on 句に入れるか、where 句に入れるかによって異なります。on 句はジョイン・テーブルの結果セットと、このジョイン・テーブルのどのローが null を供給された値を持つのかを定義します。where 句はジョイン・テーブルのどのローが結果セットに含まれるのかを定義します。

ジョイン条件で on 句または where 句のどちらを使用するかは、結果セットに何を含めるのかによって異なります。次の例は、述語を on 句または where 句のどちらに配置するのかを決定するときに役に立ちます。

外部テーブルにおける述語の制限

次のクエリは、where 句の中に外部テーブルの制限を配置します。制限は、外部ジョインの結果に適用されるので、条件が true ではないローはすべて削除します。

```
select title, titles.title_id, price, au_id
from titles left join titleauthor
on titles.title_id = titleauthor.title_id
where titles.price > $20.00

title               title_id price      au_id
------------------- -------- ---------- -----------------
But Is It User F...PC1035        22.95 238-95-7766
Computer Phobic ...PS1372        21.59 724-80-9391
Computer Phobic ...PS1372        21.59 756-30-7391
Onions, Leeks, a...TC3218        20.95 807-91-6654
(4 rows affected)
```

ここでは 4 つのローが基準を満たし、それらのローだけが結果セットに含まれます。

ただし、外部テーブルのこの制限を on 句に移動する場合、結果セットには on 句の条件を満たすローがすべて含まれます。条件を満たさない外部テーブルのローは null で拡張されます。

```
select title, titles.title_id, price, au_id
from titles left join titleauthor
on titles.title_id = titleauthor.title_id
and titles.price > $20.00
title                 title_id   price   au_id
--------------------  ---------  ------  ---------------
The Busy Executive's  BU1032     19.99   NULL
Cooking with Compute  BU1111     11.95   NULL
You Can Combat Compu  BU2075      2.99   NULL
Straight Talk About   BU7832     19.99   NULL
Silicon Valley Gastro MC2222     19.99   NULL
The Gourmet Microwave MC3021      2.99   NULL
The Psychology of Com MC3026      NULL   NULL
But Is It User Friend PC1035     22.95   238-95-7766
Secrets of Silicon Va PC8888     20.00   NULL
Net Etiquette         PC9999      NULL   NULL
Computer Phobic and   PS1372     21.59   724-80-9391
Computer Phobic and   PS1372     21.59   756-30-7391
Is Anger the Enemy?   PS2091     10.95   NULL
Life Without Fear     PS2106      7.00   NULL
Prolonged Data Depri  PS3333     19.99   NULL
Emotional Security:   PS7777      7.99   NULL
Onions, Leeks, and Ga TC3218     20.95   807-91-6654
Fifty Years in Buckin TC4203     11.95   NULL
Sushi, Anyone?        TC7777     14.99   NULL
(19 rows affected)
```

制限を on 句に移動することによって、null が供給されたローが 15 個、結果セットに追加されました。

一般に、クエリが外部テーブルに対する制限を使用し、その結果セットで制限が false であるローだけを削除する場合、制限を where 句に指定して結果セットのローを限定してください。外部テーブルの述語は、on 句の中にある場合はインデックス・キーには使用されません。

外部テーブルの制限を on 句に置くのか、where 句に置くのかということは、結局はクエリからどんな情報を返すのかによって決まります。制限が true であるローだけを結果セットに含める場合、制限は where 句に指定します。ただし、結果セットが制限を満たすかどうかに関係なく、外部テーブルのすべてのローを結果セットに含める必要がある場合、制限は on 句に指定します。

内部テーブルに対する制限

次のクエリには、where 句の中に内部テーブルの制限が含まれています。

```
select title, titles.title_id, titles.price, au_id
from titleauthor left join titles
on titles.title_id = titleauthor.title_id
where titles.price > $20.00

title            title_id   price    au_id
-------------    --------   -----    -----------
But Is It U...   PC1035     22.95    238-95-7766
Computer Ph...   PS1372     21.59    724-80-9391
Computer Ph...   PS1372     21.59    756-30-7391
Onions, Lee...   TC3218     20.95    807-91-6654
(4 rows affected)
```

where 句の制限は、ジョインが行われた後に結果セットに適用されるので、制限が true ではないローがすべて結果セットから削除されます。言い換えると、where 句はすべての null を供給された値に対しては true ではないので、それらの値を削除します。where 句に制限を配置するジョインは、事実上、内部ジョインになります。

ただし、制限は on 句に移動されると、ジョイン中に適用され、ジョイン・テーブルの生成で利用されます。この場合、結果セットは制限が true である内部テーブルのすべてのローを含み、さらに、外部テーブルのローすべてを含みます。この外部テーブルのすべてのローは、制限基準を満たさない場合には null で拡張されます。

```
select title, titles.title_id, price, au_id
from titleauthor left join titles
on titles.title_id = titleauthor.title_id
and price > $20.00

title      title_id   price           au_id
---------  ---------  -----------     -----------
NULL       NULL       NULL            172-32-1176
NULL       NULL       NULL            213-46-8915
. . .
Onions,    TC3218     20.95           807-91-6654
. . .
NULL       NULL       NULL            998-72-3567
NULL       NULL       NULL            998-72-3567
(25 rows affected)
```

この結果セットには、前述の例には含まれていなかった 21 個のローが含まれています。

一般に、クエリが内部テーブルで制限を必要とする場合 (たとえば、前述のクエリの “and price > $20.00”)、on 句に条件を入れます。これによって外部テーブルのローが保持されます。where 句に内部テーブル用の制限を指定した場合、結果セットは外部テーブルのローを含まない可能性があります。

外部テーブルに制限を指定する基準と同様、内部テーブル用の制限を on 句と where 句のどちらに指定するかは、最終的には必要な結果セットによって決まります。制限が true であるローだけを検索する場合は、制限を where 句に指定します。ただし、結果セットが制限を満たすかどうかに関係なく、外部テーブルのすべてのローを結果セットに含める必要がある場合、制限は on 句に指定します。

内部テーブルと外部テーブルの両方に含まれる制限

次のクエリにある where 句の制限は、内部テーブルと外部テーブルの両方を含んでいます。

```
select title, titles.title_id, price, price*qty, qty
from salesdetail left join titles
on titles.title_id = salesdetail.title_id
where price*qty > $30000.00


title               title_id   price  qty
-----------------   --------   -----  ---------  -----
Silicon Valley Ga   MC2222     19.99  40,619.68  2032
But Is It User Fr   PC1035     22.95  45,900.00  2000
But Is It User Fr   PC1035     22.95  45,900.00  2000
But Is It User Fr   PC1035     22.95  49,067.10  2138
Secrets of Silico   PC8888     20.00  40,000.00  2000
Prolonged Data De   PS3333     19.99  53,713.13  2687
Fifty Years in Bu   TC4203     11.95  32,265.00  2700
Fifty Years in Bu   TC4203     11.95  41,825.00  3500
(8 rows affected)
```

where 句に制限を埋め込むと、次のものが排除されます。

- 制限 “price*qty>$30000.0” が false であるロー
- price が null であるために制限 “price*qty>$30000.0” が不明であるロー

外部テーブルにある一致しないローを保持するには、on 句に制限を移動します。

```
select title, titles.title_id, price, price*qty, qty
from salesdetail left join titles
on titles.title_id = salesdetail.title_id
and price*qty > $30000.00


title               title_id   price  qty
-----------------   --------   -----  ---------  -----
```

```
NULL                NULL       NULL   NULL       75
NULL                NULL       NULL   NULL       75
. . .
Secrets of Silico   PC8888     20.00  40,000.00  2000
. . .
NULL                NULL       NULL   NULL       300
NULL                NULL       NULL   NULL       400
(116 rows affected)
```

このクエリは、結果セットに salesdetail テーブルの 116 個のローすべてを保持し、制限を満たさないローを null で拡張します。

内部テーブルと外部テーブルの両方を含む制限を指定する場所は、必要な結果セットによって決まります。制限が true であるローだけを検索する場合は、制限を where 句に指定します。ただし、制限を満たすかどうかに関係なく、外部テーブルのすべてのローを含める場合、制限を on 句に指定します。

ネストした ANSI 外部ジョイン

ネストした外部ジョインは、1 つの外部ジョインの結果セットを別の外部ジョインのテーブル参照として使用します。次に例を示します。

```
select t.title_id, title, ord_num, sd.stor_id, stor_name
from (salesdetail sd
left join titles t
on sd.title_id = t.title_id)   /*join #1*/
left join stores
on sd.stor_id = stores.stor_id /*join #2*/
title_id  title         ord_num  stor_id  stor_name
--------  ------------  -------  -------  --------------------
-----
TC3218    Onions, L...  234518   7896     Fricative Bookshop
TC7777    Sushi, An...  234518   7896     Fricative Bookshop
. . .
TC4203    Fifty Yea...  234518   6380     Eric the Read
Books
MC3021  The Gourmet...  234518   6380     Eric the Read
Books
(116 rows affected)
```

この例では、まず salesdetail テーブルと titles テーブルの間のジョイン・テーブルが論理的に生成され、その後、salesdetail.stor_id が stores.stor_id と等しい、stores テーブルのカラムとジョインされます。セマンティック的には、ジョイン内の各ネスト・レベルでジョイン・テーブルが作成され、その後、それが次のジョインに使用されます。

前述のクエリでは、最初の外部ジョインが 2 番目の外部ジョインの演算子になるために、このクエリは「左にネストした外部ジョイン」になります。

次の例は、「右にネストした外部ジョイン」を示しています。

```
select stor_name, qty, date, sd.ord_num
from salesdetail sd left join (stores /*join #1 */
left join sales on stores.stor_id = sales.stor_id) /*join #2 */
on stores.stor_id = sd.stor_id
where date > "1/1/1990"
stor_name      qty  date                 ord_num
------------   ---- -------------------  --------------------
News & Brews   200  Jun 13 1990 12:00AM  NB-3.142
News & Brews   250  Jun 13 1990 12:00AM  NB-3.142
News & Brews   345  Jun 13 1990 12:00AM  NB-3.142
. . .
Thoreau Read   1005 Mar 21 1991 12:00AM  ZZ-999-ZZZ-999-0A0
Thoreau Read   2500 Mar 21 1991 12:00AM  AB-123-DEF-425-1Z3
Thoreau Read   4000 Mar 21 1991 12:00AM  AB-123-DEF-425-1Z3
```

この例では、2 番目のジョイン (stores テーブルと sales テーブルの間) が論理的に最初に作成され、salesdetail テーブルとジョインされます。2 番目の外部ジョインが 1 番目の外部ジョイン用のテーブル参照として使われるので、このクエリは「右にネストした外部ジョイン」です。

1 番目の外部ジョイン “from salesdetail. . .” 用の on 句は、失敗した場合、2 番目の外部ジョインの stores テーブルと sales テーブルの両方に null 値を返します。

ネストした外部ジョインのカッコ

ネストした外部ジョインは、カッコのあるなしに関係なく同じ結果セットを生成します。多数の外部ジョインが含まれる大きなクエリは、カッコでジョインを構造化した方がユーザにとってより読みやすくなります。

ネストした外部ジョインでの on 句

ネストした外部ジョインでの on 句の配置は、どのジョインが論理的に最初に処理されるのかを決定します。左から右に読み取ると、最初の on 句が定義される最初のジョインになります。

この例では、最初のジョインの on 句の位置 (カッコ内) は、これが 2 番目のジョイン用のテーブル参照であることを示すので、これが 1 番目に定義され、authors テーブルとジョインされるテーブル参照を生成します。

```
select title, price, au_fname, au_lname
from (titles left join titleauthor
on titles.title_id = titleauthor.title_id)   /*join #1*/
left join authors
on titleauthor.au_id = authors.au_id      /*join #2*/
and titles.price > $15.00
title            price      au_fname      au_lname
---------------  ---------  ------------  -------------
The Busy Exe...  19.99      Marjorie      Green
The Busy Exe...  19.99      Abrahame      Bennet
. . .
Sushi, Anyon...  14.99      Burt          Gringlesby
Sushi, Anyon...  14.99      Akiko         Yokomoto
(26 rows affected)
```

ただし、on 句が違う位置にある場合、ジョインは違うシーケンスで評価されますが、それでも同じ結果セットを生成します (この例は、説明のためだけに紹介しています。ジョインされたテーブルが論理的に違う順序で生成された場合、同じ結果セットを生成することはほとんどありません)。

```
select title, price, au_fname, au_lname
from titles left join
(titleauthor left join authors
on titleauthor.au_id = authors.au_id)        /*join #2*/
on titles.title_id = titleauthor.title_id    /*join #1*/
and au_lname like"Yokomoto"

title                    price        au_fname          au_lname
----------------------   -------      -----------       -----------
The Busy Executive's     19.99        Marjorie          Green
The Busy Executive's     19.99        Abraham           Bennet
. . .
Sushi, Anyone?           14.99        Burt              Gringlesby
Sushi, Anyone?           14.99        Akiko             Yokomoto

(26 rows affected)
```

1 番目のジョインの on 句の位置 (クエリの最後の行) は、2 番目の左ジョインが 1 番目のジョインのテーブル参照であることを示しているので、これが最初に実行されます。つまり、2 番目の左ジョインの結果が titles テーブルにジョインされます。

ジョイン順依存性による外部ジョインの変換

12.0 よりも前のバージョンの Adaptive Server で作成されたほとんどすべての Transact-SQL 外部ジョインは、バージョン 12.0 以降で動作し、同じ結果セットを生成します。ただし、外部ジョイン・クエリのカテゴリには、最適化中に選択されるジョイン順に結果セットが依存するものがあります。これらのクエリは、クエリのどこで述語が評価されるのかによって、Adaptive Server の新しいバージョンを使って発行されるときに違った結果セットを生成する可能性があります。返される結果セットは、ジョインに述語を割り当てるために ANSI 規則によって判断されます。

述語は、参照するすべてのテーブルが処理されるまで評価できません。つまり、次のクエリで、述語 “and titles.price > 20” は titles テーブルが処理されるまで評価できません。

```
select title, price, au_ord
from titles, titleauthor
where titles.title_id *= titleauthor.title_id
and titles.price > 20
```

12.0 より前のバージョンの Adaptive Server にある述語は、次のセマンティックに従って評価されていました。

- 外部ジョインの内部テーブルで述語が評価された場合、述語は on 句のセマンティックを持っていた。
- すべての外部ジョインに対して外部にあるテーブルで述語が評価された場合、または、述語がジョイン順に依存しない場合、述語は where 句のセマンティックを持っていた。

注意 Adaptive Server を運用環境で実行する前に、トレース・フラグ 4413 で Adaptive Server を起動し、バージョン 12.0 よりも前のバージョンでジョイン順に依存していた可能性のあるクエリをすべて実行して確認してください。ジョイン順依存クエリを実行すると、次のようなメッセージが表示されます。

```
Warning: The results of the statement on line %d are join-
order independent.Results may differ on pre-12.0 releases,
where the query is potentially join-order dependent.
```

12.0 よりも前のバージョンで生成された、ジョイン順クエリの結果セットに対してアプリケーションが持っている依存性を解決します。

一般に、述語は通常は次のものを参照するだけなので、ジョイン順依存クエリでは何の問題も出ません。

- where 句のセマンティックを使って評価される外部テーブル
- on 句のセマンティックを使って評価される内部テーブル
- 内部テーブルが依存する内部テーブル

これらは、ジョイン順依存の外部ジョインは生成しません。ただし、次の特徴のいずれかを持つ Transact-SQL クエリは、ANSI 外部ジョインに変換された後に異なる結果セットを生成する可能性があります。

- or　文を含み、外部ジョインの内部テーブルと、この内部テーブルが依存しないもう 1 つのテーブルを参照する述語
- 内部テーブルのジョイン順依存にはないテーブルと同じ述語に参照される内部テーブル属性
- 相関参照としてサブクエリに参照される内部テーブル

次の例は、ジョイン順依存性のある Transact-SQL クエリを ANSI 外部ジョイン・クエリに変換することを示しています。

このクエリでは、外部ジョインが titleauthor テーブルと titles テーブルの両方を参照し、authors テーブルは次の 3 つのジョイン順に従ってこれらのテーブルとジョインできるので、このクエリはジョイン順には依存しません。

- authors、titleauthors、titles (on 句の一部)
- titleauthors、authors、titles (on 句の一部)
- titleauthors、titles、authors (where 句の一部)

```
select title, price, authors.au_id, au_lname
from titles, titleauthor, authors
where titles.title_id =* titleauthor.title_id
and titleauthor.au_id = authors.au_id
and (titles.price is null or authors.postalcode = '94001')
```

そして、この句は、次のメッセージを生成します。

```
Warning: The results of the statement on line 1 are join-order independent.Results may
differ on pre-12.0 releases, where the query is potentially join-order dependent.Use trace
flag 4413 to suppress this warning message.
```

次に示すのは ANSI と同等のクエリです。

```
select title, price, authors.au_id, au_lname
from titles right join
(titleauthor inner join authors
on titleauthor.au_id = authors.au_id)
on titles.title_id = titleauthor.title_id
where (titles.price is null or authors.postalcode = '94001')
```

このクエリは、1 つ前の例と同じ理由でジョイン順に依存します。

```
select title, au_fname, au_lname, titleauthor.au_id, price
from titles, titleauthor, authors
where authors.au_id *= titleauthor.au_id
and titleauthor.au_ord*titles.price > 40
```

次に示すのは ANSI と同等のクエリです。

```
select title, au_fname, au_lname, titleauthor.au_id, price
from titles,(authors left join titleauthor
on titleauthor.au_id = authors.au_id)
where titleauthor.au_ord*titles.price > 40
```

Transact-SQL 外部ジョイン

Transact-SQL は左と右の両方の外部ジョインの構文を含みます。「左外部ジョイン」、つまり、***=** は、文の制限を満たす 1 番目のテーブルのローをすべて選択します。2 番目のテーブルは、ジョイン条件に一致がある場合、値を生成します。一致しない場合、2 番目のテーブルは null 値を生成します。

たとえば、次の左外部ジョインは、すべての作家をリストして、その都市に出版社があれば検出します。

```
select au_fname, au_lname, pub_name
from authors, publishers
where authors.city *= publishers.city
```

「右外部ジョイン」、つまり、=* は、文の制限を満たす 2 番目のテーブルのローをすべて選択します。1 番目のテーブルは、ジョイン条件に一致がある場合、値を生成します。一致しない場合、1 番目のテーブルは null 値を生成します。

注意 having 句には Transact-SQL の外部ジョインを含めることはできません。

テーブルは、外部ジョインの内部または外部のいずれかのメンバです。ジョイン演算子が *= の場合、2 番目のテーブルが内部テーブルです。ジョイン演算子が =* の場合、1 番目のテーブルが内部テーブルです。内部テーブルのカラムは、外部ジョインで使用するように、定数と比較できます。たとえば、どの title が 4000 部よりも多く売れたのかを検索するには、次のように指定します。

```
select qty, title from salesdetail, titles
where qty > 4000
and titles.title_id *= salesdetail.title_id
```

ただし、外部ジョインの中の内部テーブルは、通常のジョイン句には指定できません。

以前の例で、出版社と同じ都市に住む作家の名前を検索して、Abraham Bennet と Cheryl Carson という 2 つの名前を返すジョインを使いました。出版社が同じ都市にあるかどうかに関係なく、結果にすべての作家を含めるには、外部ジョインを使います。次に示すのは、クエリとその外部ジョインの結果です。

```
select au_fname, au_lname, pub_name
from authors, publishers
where authors.city *= publishers.city
```

```
au_fname     au_lname         pub_name
---------    --------------   ---------------
Johnson      White            NULL
Marjorie     Green            NULL
Cheryl       Carson           Algodata Infosystems
Michael      O'Leary          NULL
Dick         Straight         NULL
Meander      Smith            NULL
Abraham      Bennet           Algodata Infosystems
Ann          Dull             NULL
Burt         Gringlesby       NULL
Chastity     Locksley         NULL
Morningstar  Greene           NULL
Reginald     Blotche-Halls    NULL
Akiko        Yokomoto         NULL
Innes        del Castillo     NULL
Michel       DeFrance         NULL
Dirk         Stringer         NULL
Stearns      MacFeather       NULL
Livia        Karsen           NULL
Sylvia       Panteley         NULL
Sheryl       Hunter           NULL
```

```
Heather      McBadden         NULL
Anne         Ringer           NULL
Albert       Ringer           NULL

(23 rows affected)
```

比較演算子 *= は通常のジョインと外部ジョインを区別するものです。「左」外部ジョインは、publishers テーブルの city カラムに一致があるかどうかに関係なく、結果に authors テーブルのすべてのローを含めます。結果からわかるように、リストされている作家のほとんどにはデータの一致がないので、これらのローは pub_name カラムに null を含んでいることがわかります。

「右」外部ジョインは、比較演算子 =* で指定されます。これは、2 番目のテーブルにあるすべてのローが、1 番目のテーブルに一致するデータがあるかどうかに関係なく、結果に含まれることを示しています。

前に示された外部ジョイン・クエリでこの演算子を置き換えると、次の結果になります。

```
select au_fname, au_lname, pub_name
from authors, publishers
where authors.city =* publishers.city
au_fname     au_lname    pub_name
---------    ---------   ---------------
NULL         NULL        New Age Books
NULL         NULL        Binnet & Hardley
Cheryl       Carson      Algodata Infosystems
Abraham      Bennet      Algodata Infosystems

(4 rows affected)
```

外部ジョインは定数と比較することによってさらに制限することができます。これは、検索したい値を正確に限定することができ、指定値を含まないローをリストするために外部ジョインを使えることを意味しています。最初に等価ジョインを実行してから、それを外部ジョインと比較してみます。たとえば、書店で 500 部を超える販売数のあったタイトルを検索するには、次のクエリを使用します。

```
select distinct salesdetail.stor_id, title
from titles, salesdetail
where qty > 500
and salesdetail.title_id = titles.title_id
stor_id
title
------- ----------------------------------------
5023    Sushi, Anyone?
5023    Is Anger the Enemy?
5023    The Gourmet Microwave
5023    But Is It User Friendly?
5023    Secrets of Silicon Valley
5023    Straight Talk About Computers
5023    You Can Combat Computer Stress!
```

```
5023    Silicon Valley Gastronomic Treats
5023    Emotional Security: A New Algorithm
5023    The Busy Executive's Database Guide
5023    Fifty Years in Buckingham Palace Kitchens
5023    Prolonged Data Deprivation: Four Case Studies
5023    Cooking with Computers: Surreptitious Balance Sheets
7067    Fifty Years in Buckingham Palace Kitchens

(14 rows affected)
```

また、次の外部ジョイン・クエリを使用して、書店で 500 部を超える販売数がなかったタイトルを示すことができます。

```
select distinct salesdetail.stor_id, title
from titles, salesdetail
where qty > 500
and salesdetail.title_id =* titles.title_id
```

```
stor_id     title

-------   ----------------------------------------
NULL    Net Etiquette
NULL    Life Without Fear
5023    Sushi, Anyone?
5023    Is Anger the Enemy?
5023    The Gourmet Microwave
5023    But Is It User Friendly?
5023    Secrets of Silicon Valley
5023    Straight Talk About Computers
NULL    The Psychology of Computer Cooking
5023    You Can Combat Computer Stress!
5023    Silicon Valley Gastronomic Treats
5023    Emotional Security: A New Algorithm
5023    The Busy Executive's Database Guide
5023    Fifty Years in Buckingham Palace Kitchens
7067    Fifty Years in Buckingham Palace Kitchens
5023    Prolonged Data Deprivation: Four Case Studies
5023    Cooking with Computers: Surreptitious Balance Sheets
NULL    Computer Phobic and Non-Phobic Individuals:
        Behavior Variations
NULL    Onions, Leeks, and Garlic: Cooking Secrets of the
        Mediterranean

(19 rows affected)
```

内部テーブルは簡単な句で制限できます。次の例は、出版社と同じ都市に住む作家をリストしたものですが、通常は出版社と同じ都市に住んでいる作家としてリストされる作家 Cheryl Carson は除いています。

```
select au_fname, au_lname, pub_name
from authors, publishers
where authors.city =* publishers.city
and authors.au_lname != "Carson"
```

```
au_fname     au_lname    pub_name
---------    ---------   ---------------
NULL         NULL        New Age Books
NULL         NULL        Binnet & Hardley
Abraham      Bennet      Algodata Infosystems

(3 rows affected)
```

外部ジョインと集合拡張カラムの使用

外部ジョインと集合拡張カラムを一緒に使用する場合、集合拡張カラムが外部ジョインの内部テーブルのカラムであるときに、クエリの結果セットと外部ジョインの結果セットが等しくなります。

外部ジョインは、Sybase の外部ジョイン演算子である =* または *= を使用して 2 つのテーブルのカラムを接続します。これらの記号は、Transact-SQL の拡張機能構文です。これらは ANSI SQL の記号ではなく、「外部ジョイン」は Transact-SQL のキーワードではありません。この項では、Sybase の構文だけについて説明します。

アスタリスクの側で指定されるカラムは、外部ジョインで使用する外部テーブルの外部カラムです。

集合拡張カラムでは集合関数 (max、min) を使用しますが、クエリの group by 句には含めません。

たとえば、null 入力ローが結果に含まれる外部ジョインを作成するには、次のように入力します。

```
select publishers.pub_id, titles.price
from publishers, titles
where publishers.pub_id *= titles.pub_id
and titles.price > 20.00
pub_id price

------ ----------------- ------
0736 NULL
0877 20.95
0877 21.59
1389 22.95

(4 rows affected)
```

同様に、null 入力ローが結果に含まれる外部ジョインと集合カラムを作成するには、次のように入力します。

```
select publishers.pub_id, max(titles.price)
from publishers, titles
where publishers.pub_id *= titles.pub_id
and titles.price > 20.00
group by publishers.pub_id
pub_id
```

```
--------------- --------
0736 NULL
0877 21.59
1389 22.95
(3 rows affected)
```

null 入力ローが結果に含まれる、集合拡張カラムを持つ外部ジョインと集合カラムを作成するには、次のように入力します。

```
select publishers.pub_id, titles.title_id,
max(titles.price)
from publishers, titles
where publishers.pub_id *= titles.pub_id
and titles.price > 20.00
group by publishers.pub_id

--------------- --------
....
(54 rows affected)
```

再配置ジョイン

ローカル・テーブルとリモート・テーブル間のジョインをリモート・サーバに再配置できます。リモート・システムでの再配置ジョインは、ローカル・テーブルを参照する、動的に作成されたプロシキ・テーブルで実行されます。これにより、ネットワーク・トラフィックが大量に発生することを回避できます。

再配置ジョインの使用

ローカル・テーブル ls1 とリモート・テーブル rl1 間のジョインによって、次のクエリがリモート・サーバに送信されます。

```
select a,b,c
from localserver.testdb.dbo.ls1 t1, rl1 t2
where t1.a = t2.a
```

リモート・サーバに送信される文には、ローカル・システム上のローカル・テーブルへの完全に修飾された参照が含まれています。リモート・サーバは、このテーブルのテンポラリ・プロシキ・テーブルの定義を動的に作成するか、マッピングが一致する既存のプロシキ・テーブルを使用するかします。その後、リモート・サーバはこのジョインを実行し、ローカル・サーバに戻された結果を返します。

『パフォーマンス&チューニング・シリーズ：クエリ処理と抽象プラン』を参照してください。

再配置ジョイン

厳密には、再配置ジョインは関連するリモート・サーバごとに有効にする必要があります。リモート・サーバは、コンポーネント統合サービス (CIS) によってローカル・サーバと接続できなければなりません。

再配置ジョインを設定するには、次の手順に従います。

1 sp_serveroption を使用して、再配置ジョインがリモート・サーバに送信されるようにします。

```
sp_serveroption servername, "relocated joins",true
```

2 リモート・サーバで次の項目を確認します。

- リモート・サーバにローカル・サーバ用のインタフェース・エントリがある。
- sysservers エントリがある (sp_addserver によって追加)。
- 外部ログインが設定されている。

3 動的に作成されたプロシキ・テーブルを使用する場合は、再配置ジョインの受け取り時に tempdb でプロシキ・テーブルが作成されます。ddl in tran を有効にして、tempdb でプロシキ・テーブルが受け入れられるようにします。

```
sp_dboption tempdb,"ddl in tran",true
```

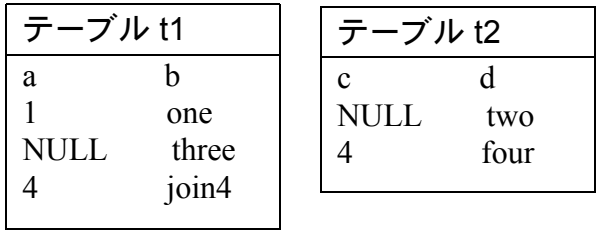
null 値がジョインに与える影響

ジョインされるテーブルまたはビューにある null 値は互いに一致することはありません。bit カラムは null 値を許さないので、内部テーブルにある bit カラムに一致がないと、外部ジョインには 0 という値が表示されます。

null と他の任意の値のジョインの結果は null です。null 値は不定の値または適用できない値を表すので、Transact-SQL には 1 つの不明の値が別の値と一致するとみなす根拠がありません。

外部ジョインを使用する場合に限り、ジョインされるテーブルの 1 つにあるカラムに null 値が存在するかどうかを検出することができます。図 4-1 では、各テーブルにジョインに参加するカラムに null 値があります。左外部ジョインは 1 番目のテーブルに null 値を表示します。

図 4-1: 外部ジョインの null 値



テーブル t1

a	b
1	one
NULL	three
4	join4

テーブル t2

c	d
NULL	two
4	four

テーブル t1 には 2 つのカラム “a” および “b” があります。テーブル t2 には 2 つのカラム “c” および “d” があります。以下、同様です。次に示すのが、左外部ジョインです。

```
select *
from t1, t2
where a *= c

a            b       c            d
-----------  ------  -----------  ------
           1  one            NULL  NULL
        NULL  three          NULL  NULL
           4  join4             4  four

(3 rows affected)
```

結果を見ると、データ内に null があるのか、ジョインの失敗を示す null なのかを見分けることは難しくなります。ジョインされるデータに null 値が存在する場合、通常は標準のジョインを使用して結果に null を表示しないようにします。

ジョインするテーブル・カラムの決定

sp_helpjoins は、ジョインの候補になる 2 つのテーブルやビューにあるカラムをリストします。

```
sp_helpjoins table1, table2
```

たとえば、titleauthor と titles の間でジョインの候補になるカラムを検索するには、次を使用します。

```
sp_helpjoins titleauthor, titles
first_pair

------------------------------ --------------------
title_id                       title_id
```

sp_helpjoins が表示するカラムの組は 2 つのソースからのものです。まず、sp_helpjoins は現在のデータベースにある syskeys テーブルをチェックして、sp_foreignkey でこの 2 つのテーブルに外部キーが定義されているかどうかを確認してから、sp_commonkey でこの 2 つのテーブルに共通キーが定義されているかどうかを確認します。共通キーがない場合、このプロシージャは、ジョインするのに最適なキーを指定するための、制限の少ない基準を適用します。プロシージャは同じユーザ・データ型のキーをチェックして、それが失敗した場合には、同じ名前とデータ型のカラムをチェックします。

『リファレンス・マニュアル：プロシージャ』を参照してください。

第 5 章

サブクエリ：他のクエリ内でのクエリの使用

「サブクエリ」は、別の select、insert、update、または delete 文の内部、条件文の内部、または、別のサブクエリの内部にネストされている select 文です。

サブクエリはジョイン演算としても表すことができます。「第 4 章 ジョイン：複数テーブルからのデータの検索」を参照してください。

サブクエリの動作

サブクエリは、「内部クエリ」とも呼ばれ、別の SQL 文の where 句または having 句の内部、または文の select リスト内に指定します。サブクエリは、別のクエリの結果として表されるクエリ要求を扱うために使うことができます。サブクエリを含む文は、サブクエリの select リストの評価に基づいて、1 つのテーブルのローで作用します。このリストは、外部クエリとして同じテーブルを参照することも、別のテーブルを参照することもできます。Transact-SQL では、サブクエリが単一の値を返す場合には、式を使用できるほぼすべての場所にサブクエリを使用できます。case 式の中でも、サブクエリが使用できます。

たとえば、次のサブクエリは印税分配が 75 ドルを超えるすべての作家の名前をリストします。

```
select au_fname, au_lname
from authors
where au_id in
   (select au_id
    from titleauthor
    where royaltyper > 75)
```

1 つ以上のサブクエリを含む select 文は、「ネストされたクエリ」または「ネストされた **select** 文」と呼ばれることもあります。

値を何も返さないサブクエリの結果は null です。サブクエリが null を返した場合、そのクエリは失敗します。

『リファレンス・マニュアル：コマンド』を参照してください。

サブクエリの制限事項

サブクエリには、次に示す制限事項があります。

- *subquery_select_list* は、exists サブクエリ内を除いて、1 つのカラム名だけで構成される必要がある。exists サブクエリの場合は、通常、単一のカラム名の代わりにアスタリスク (*) が使われる。exists サブクエリではないネストされた select 文にアスタリスク (*) を使用できる。

  複数のカラム名は指定しないこと。カラム名が属しているテーブルやビューがあいまいな場合には、必ずカラム名をテーブルやビューで修飾する。

- サブクエリは、外部の select、insert、update、または delete 文の where 句や having 句の内部、別のサブクエリの内部、または、select リストの内部にネストすることができる。その他に、サブクエリを含む多くの文をジョインとして作成することができる。Adaptive Server はそのような文をジョインとして処理する。

- Transact-SQL では、サブクエリが単一の値を返す場合には、式を使用できるほぼすべての場所にサブクエリを使用できる。SQL 抽出テーブルは、サブクエリがどこで使用されているかにかかわらず、サブクエリの from 句で使用できる。

- order by、group by、または compute by リストの中ではサブクエリを使用できない。

- サブクエリには for browse 句を使用できない。

- サブクエリ内の抽出テーブル式の一部である場合を除き、サブクエリに union 句を含めることはできない。SQL 抽出テーブルの使用方法については、「第 9 章 SQL 抽出テーブル」 を参照。

- 比較演算子とともに指定された内部サブクエリの select リストに含めることができるのは 1 つの式またはカラム名だけであり、サブクエリは単一の値を返す必要がある。外部の文の where 句で名前を付けたカラムは、サブクエリの select リストで名前を付けたジョインと互換である必要がある。

- text、unitex、image データ型をサブクエリに含めることができない。

- サブクエリは内部でその結果を操作できない。つまり、サブクエリは order by 句、compute 句、または into キーワードを含めることができない。

- 相関する (繰り返しの) サブクエリは、declare cursor によって定義された更新可能なカーソルの select 句の中では使用できない。

- ネスト・レベルの制限は 50 以内である。

- union のそれぞれの側にあるサブクエリの最大数は 50 である。

- サブクエリが外部クエリの having 句の中にあり、集約値が外部クエリの from 句にあるテーブルのカラムである場合のみ、そのサブクエリの where 句に集合関数を指定することができる。
- サブクエリの結果式には、式に対する制限と同じ制限が適用される。式の最大長は 16K である。『リファレンス・マニュアル：ビルディング・ブロック』の「第 4 章 式、識別子、およびワイルドカード文字」を参照。

サブクエリの使用例

『traight Talk About Computers』と同じ価格の本を検索するには、まず、『Straight Talk』の価格を検索します。

```
select price
from titles
where title = "Straight Talk About Computers"

price
-------------
       $19.99

(1 row affected)
```

この結果を 2 番目のクエリに使って、『Straight Talk』と同じ価格の本をすべて検索します。

```
select title, price
from titles
where price = $19.99

title                                          price
---------------------------------------------  -----
The Busy Executive’s Database Guide            19.99
Straight Talk About Computers                  19.99
Silicon Valley Gastronomic Treats              19.99
Prolonged Data Deprivation: Four Case Studies  19.99
```

サブクエリを使用すると、同じ結果をたった 1 つの手順で得ることができます。

```
select title, price
from titles
where price =
     (select price
    from titles
    where title = "Straight Talk About Computers")

title                                          price
---------------------------------------        -----
The Busy Executive’s Database Guide            19.99
Straight Talk About Computers                  19.99
Silicon Valley Gastronomic Treats              19.99
Prolonged Data Deprivation: Four Case Studies  19.99
```

カラム名の修飾

文の中でのカラム名は、同じレベルの from 句に参照されるテーブルによって暗黙のうちに修飾されます。次の例では、テーブル名 publishers は暗黙的に外部クエリの where 句にある pub_id カラムを修飾します。サブクエリの select リストにある pub_id への参照は、サブクエリの from 句、つまり、titles テーブルによって修飾されます。

```
select pub_name
from publishers
where pub_id in
   (select pub_id
    from titles
    where type = "business")
```

次は、暗黙の修飾を記述したときにクエリがどのようになるかを示します。

```
select pub_name
from publishers
where publishers.pub_id in
   (select titles.pub_id
    from titles
    where type = "business")
```

テーブル名を明示的に指定することは間違いではありません。また、テーブル名の暗黙の修飾は、いつでも明示的な修飾で上書きできます。

相関名によるサブクエリ

自分自身にジョインするテーブルが 2 つの異なる役割を果たすため、セルフジョインにはテーブル相関名が必要になります (「第 4 章 ジョイン：複数テーブルからのデータの検索」を参照)。相関名は、内部クエリと外部クエリの両方で同じテーブルを参照するネストされたクエリに使うことができます。

たとえば、Livia Karsen と同じ都市に住む作家を検索するには、次のようにします。

```
select au1.au_lname, au1.au_fname, au1.city
from authors au1
where au1.city in
   (select au2.city
    from authors au2
    where au2.au_fname = "Livia"
    and au2.au_lname = "Karsen")

au_lname     au_fname     city
-----------  ---------    -------
Green        Marjorie     Oakland
Straight     Dick         Oakland
Stringer     Dirk         Oakland
MacFeather   Stearns      Oakland
Karsen       Livia         Oakland
```

明示的な相関名によって、サブクエリでの authors への参照は外部クエリでの authors への参照と同じ意味ではないことが明らかになります。

明示的な相関を行わないと、サブクエリは次のようになります。

```
select au_lname, au_fname, city
from authors
where city in
   (select city
    from authors
    where au_fname = "Livia"
    and au_lname = "Karsen")
```

代わりに、上記クエリを、セルフジョインとして、同じテーブルを参照する外部クエリとサブクエリを使用した文で記述します。

```
select au1.au_lname, au1.au_fname, au1.city
from authors au1, authors au2
where au1.city = au2.city
and au2.au_lname = "Karsen"
and au2.au_fname = "Livia"
```

ジョインとして再度指定されたサブクエリは、同じ順で結果を返さない場合もあり、ジョインは重複を削除するために distinct キーワードを必要とする可能性もあります。

ネストの複数のレベル

サブクエリには 1 つ以上のサブクエリを含めることができます。1 つの文の中では最大 50 個のサブクエリをネストすることができます。

たとえば、少なくとも 1 冊は一般向けのコンピュータの本を執筆している作家を検索するには、次のように入力します。

```
select au_lname, au_fname
from authors
where au_id in
   (select au_id
    from titleauthor
    where title_id in
      (select title_id
        from titles
        where type = "popular_comp") )

au_lname               au_fname
---------------------- ------------
Carson                 Cheryl
Dull                   Ann
Locksley               Chastity
Hunter                 Sheryl

(4 rows affected)
```

最も外側のクエリは、すべての作家名を選択しています。次のクエリは作家の ID を検索し、最も内側のクエリはタイトル ID 番号である PC1035、PC8888、PC9999 を返します。

また、このクエリをジョインとして表現することもできます。

```
select au_lname, au_fname
from authors, titles, titleauthor
where authors.au_id = titleauthor.au_id
and titles.title_id = titleauthor.title_id
and type = "popular_comp"
```

ネストされた *select* 文でのアスタリスクの使用

アスタリスクが次の条件である限り、exists サブクエリではないネストされた select 文にアスタリスク (*) を使用できます。

- select 文の唯一の項目である
- ネストされたクエリの 1 つのテーブルカラムを解決する

さらに、次の操作ができます。

- *qualifier*.* フォーマット (ここで、*qualifier* は from 句内の 1 つのテーブル) を使用して、ネストされたクエリ内で選択するカラムを特定のテーブルに属するカラムのみに制限できます。
- group by 句を含むネストされたクエリにアスタリスクを使用できます。

アスタリスクがネストされたクエリの 1 つのテーブルカラムに解決される場合、このクエリは 1 つのテーブルカラムを明示的に使用する場合と同様になります。

t2 に含まれるカラムが 1 つのみであるため、これはネストされた有効なクエリです。

```
1> create table t1(c1 int, c2 int)
2> create table t2(c1 int)
3> go
1> select * from t1 where c1 in (select * from t2)
2> go
```

ネストされた select 文は次のものと同等です。

```
1> select * t1 where c1 in (select c1 from t2)
2> go
```

15.7 よりも前のバージョンの Adaptive Server では、ネストされた exists サブクエリでしかアスタリスクを使用できませんでした。

テーブル名修飾子の使用

qualifier.* (修飾子 <ピリオド> アスタリスク) の形式でアスタリスクを使用すると、次のように、指定されたテーブルにあるカラムのみを選択できます。

```
1> create table t1(c1 int, c2 int)
2> go
1> create table t2(c1 int)
2> go
1> select * from t1
2> where c1 in (select t2.* from t1, t2)
3> go
```

ネストされた select 文は次のものと同等です。

```
1> select * from t1
2> where c1 in (select t2.c1 from t1, t2)
3> go
```

group by でのネストされたクエリの使用

次のように、group-by テーブルが単一カラム テーブルである限り、ネストされた group by クエリにアスタリスクを使用できます。

```
1> select * from t1
2> where c1 in (select * from t2 group by c1)
3> go
```

ネストした group by クエリの例は次のものと同等です。

```
1> select * from t1
2> where c1 in (select c1 from t2 group by c1)
3> go
```

例

例 1　sales のない、または discount が 10 よりも大きい stores から discount を削除します。

```
create view store_with_nosales(stor_id)
as
select stores.stor_id
from stores left join sales
     on stores.stor_id = sales.stor_id
where sales.stor_id IS NULL
go

delete from discounts
where (stor_id in (select *
          from store_with_nosales)
     or discount > 10.0)
go
```

例 2　stores と sales のジョインに複数のカラムがあるため、エラーを返します。

```
create view store_with_nosales(stor_id)
as
select stores.stor_id
from stores left join sales
     on stores.stor_id = sales.stor_id
where (stor_id in (select *
       from stores left join sales
            on stores.stor_id = sales.stor_id
        where sales.stor_id IS NULL)
  or discount > 10.0)
go

delete from discounts
where (stor_id in (select *
          from store_with_nosales)
     or discount > 10.0)
go

Msg 299, Level 16, State 1:
Line 1:
The symbol '*' can only be used for a subquery select
list when the subquery is introduced with EXISTS or NOT
EXISTS or the subquery references a single table and
column.
```

使用法

Adaptive Server は、新しいストアド・プロシージャ、ビュー、トリガを保存する前に、自動的にクエリ内のアスタリスクを実際のカラム名に置き換えます。この置き換えは、テーブルを変更してカラムを追加する場合にも存続します。Adaptive Server では複数のカラムを使用できませんが、アスタリスクの置き換えにより、追加のカラムが導入されます。この不正な動作は、テキストを削除して再作成するまで継続します。次に例を示します。

```
1> create table t1(c1 int, c2 int)
2> go
1> create table t2(c1 int)
2> go
1> create proc p1
2> as
3> select * from t1 where c1 in (select * from t2)
4> go
1> exec p1
2> go
 c1          c2
 ----------- -----------

(0 rows affected)
(return status = 0)
```

```
1> sp_helptext p1
2> go
# Lines of Text
 ---------------
               2

(1 row affected)
text
------------------------------------------------------------------
------------------------------------------------
create proc p1
as/* Adaptive Server は、次の文ですべての '*' 要素を拡張しています。
*/

select t1.c1, t1.c2
    from t1 where c1 in (select t2.c1 from t2)

(2 rows affected)
(return status = 0)
1> alter table t2 add c2 int null
2> go
1> exec p1
2> go
 c1          c2
 ----------- -----------

(0 rows affected)
(return status = 0)

1> exec p1 with recompile
2> go
 c1          c2
 ----------- -----------

(0 rows affected)
(return status = 0)
1> drop proc p1
2> go
1> create proc p1
2> as
3> select * from t1 where c1 in (select * from t2)
4> go
Msg 299, Level 16, State 1:
Procedure 'p1', Line 4:
The symbol '*' can only be used for a non-EXISTS
subquery select list when the subquery is on a single
table with a single column.
```

Adaptive Server は、アスタリスクが単一カラムを解決することを予測し、アスタリスクを変換した後に複数のカラムが発生した場合にエラーを生成します。

update 文、*delete* 文、および *insert* 文でのサブクエリ

サブクエリは、select 文の中と同様、update、delete、insert 文の中でもネストすることができます。

注意 この項のサンプル・クエリを実行すると、pubs2 データベースが変更されます。これらのクエリを実行した後に元の pubs2 データベースが必要となった場合は、システム管理者に依頼して pubs2 データベースを再ロードしてもらってください。

次のクエリは、New Age Books によって出版されたすべての本の価格を倍にします。この文は titles テーブルを更新し、サブクエリは publishers テーブルを参照します。

```
update titles
set price = price * 2
where pub_id in
   (select pub_id
    from publishers
    where pub_name = "New Age Books")
```

ジョインを使用したこれと等価な update 文は次のようになります。

```
update titles
set price = price * 2
from titles, publishers
where titles.pub_id = publishers.pub_id
and pub_name = "New Age Books"
```

次のネストした select 文ではビジネス書販売のすべてのレコードを削除します。

```
delete salesdetail
where title_id in
   (select title_id
    from titles
    where type = "business")
```

ジョインを使ったこれと等価な delete 文は次のようになります。

```
delete salesdetail
from salesdetail, titles
where salesdetail.title_id = titles.title_id
and type = "business"
```

条件文のサブクエリ

サブクエリは、条件文で使うことができます。ビジネス書の販売についてのすべてのレコードを削除する前述のサブクエリは、次の例のように書き直して、レコードをチェックした後で削除するようにします。

```
if exists (select title_id
    from titles
    where type = "business")
begin
    delete salesdetail
    where title_id in
      (select title_id
      from titles
      where type = "business")
end
```

式の代わりとしてのサブクエリ

Transact-SQL では、select、update、insert、または delete 文の式を使用できるほとんどすべての場所で式をサブクエリに置き換えることができます。たとえば、サブクエリは外部ジョインの内部テーブルにあるカラムと比較することができます。

サブクエリは、order by リストの中や、insert 文にある values リストの式としては使用できません。

次の文は、カリフォルニア州に住む作家によって書かれ、州内で出版された本のタイトルと種類を検索する方法を示しています。

```
select title, type
from titles
where title in
   (select title
    from titles, titleauthor, authors
    where titles.title_id = titleauthor.title_id
    and titleauthor.au_id = authors.au_id
    and authors.state = "CA")
and title in
   (select title
    from titles, publishers
    where titles.pub_id = publishers.pub_id
    and publishers.state = "CA")

title                                type
-----------------------------------  ----------
The Busy Executive's Database Guide  business
Cooking with Computers:
     Surreptitious Balance Sheets    business
Straight Talk About Computers        business
But Is It User Friendly?
```

```
Secrets of Silicon Valley             popular_comp
Net Etiquette                         popular_comp

(6 rows affected)
```

次の文は、5000 部を超えて販売された本のタイトルを選択し、その価格と、最も高い本の価格をリストします。

```
select title, price,
    (select max(price) from titles)
    from titles
    where total_sales > 5000
```

```
title                                price
-----------------------------------  -----  ------
You Can Combat Computer Stress!       2.99   22.95
The Gourmet Microwave                 2.99   22.95
But Is It User Friendly?             22.95   22.95
Fifty Years in Buckingham Palace
    Kitchens                         11.95   22.95

(4 rows affected)
```

サブクエリのタイプ

サブクエリには基本的に 2 つのタイプがあります。

- 「式サブクエリ」は、修飾されていない比較演算子とともに指定され、単一の値を返す必要があり、また、SQL で式が許可されているほとんどどこでも使用できる。
- 「限定述語サブクエリ」は、in とともに、あるいは any または all によって修飾された比較演算子とともに指定されたリストに対して動作する。限定述語サブクエリは、0 以上の値を返します。このタイプは、exists とともに指定される (サブクエリがローを生成するかどうかをチェックする) 存在テストとしても使用される。

どちらのタイプも、非相関サブクエリ、または相関サブクエリ (繰り返し) にできます。

- 「非相関サブクエリ」は、それが独立したクエリであるかのように評価できる。概念的には、サブクエリの結果はメインの文、または外部クエリに置き換えられる。これは、Adaptive Server が実際にサブクエリで文を処理する方法ではない。非相関サブクエリは、ジョインとして記述することも可能で、Adaptive Server ではジョインとして処理される。
- 「相関サブクエリ」は独立したクエリとしては評価できないが、外部クエリの from リストにリストされたテーブルのカラムを参照できる。相関サブクエリについては、この章の最後で詳細を説明する。

式サブクエリ

式サブクエリには次のものがあります。

- (in で指定される) select リスト内のサブクエリ
- 比較演算子 (=、!=、>、>=、<、<=) によって結合された where または having 句の中のサブクエリ

式サブクエリの一般的な形式は次のようになります。

[select、insert、update、delete 文またはサブクエリの始まり]

```
where expression comparison_operator (subquery)
```

[select、insert、update、delete 文またはサブクエリの終わり]

式は、カラム名、定数、および算術演算子やビット処理演算子によって結合された関数の任意の組み合わせ、または、1 つのサブクエリから構成されます。

comparison_operator は次のいずれかです。

演算子	意味
=	等しい
>	より大きい
<	より小さい
>=	以上
<=	以下
!=	等しくない
<>	等しくない
!>	より大きくない
!<	より小さくない

外部文の where または having 句にカラム名を使う場合、*subquery_select_list* 内のカラム名はその where 句か having 句の中のカラム名とジョインで置き換えられることを確認してください。

修飾されていない比較演算子 (つまり、後ろに any や all がない比較演算子) で指定されるサブクエリは、単一の値を返す必要があります。そのようなサブクエリが複数の値を返す場合、Adaptive Server はエラー・メッセージを返します。

たとえば、出版社それぞれが 1 つの都市にだけにあると仮定します。Algodata Infosystems 社のある都市に住んでいる作家の名前を検索するには、比較演算子 = で指定されるサブクエリのある、次のような文を作成します。

```
select au_lname, au_fname
from authors
where city =
   (select city
    from publishers
    where pub_name = "Algodata Infosystems")

au_lname        au_fname
```

```
-------------- --------------
Carson         Cheryl
Bennet         Abraham
```

単一の値を保証するためのスカラ集合関数の使用

修飾されていない比較演算子で指定されるサブクエリには、多くの場合、単一の値を返すスカラ集合関数を含みます。

たとえば、次の文は現在の最低価格よりも高い価格の本のタイトルを検索します。

```
select title
from titles
where price >
   (select min(price)
    from titles)

title
--------------------------------------------------
The Busy Executive's Database Guide
Cooking with Computers: Surreptitious Balance
     Sheets
Straight Talk About Computers
Silicon Valley Gastronomic Treats
But Is It User Friendly?
Secrets of Silicon Valley
Computer Phobic and Non-Phobic Individuals:
     Behavior Variations
Is Anger the Enemy?
Life Without Fear
Prolonged Data Deprivation: Four Case Studies
Emotional Security: A New Algorithm
Onions, Leeks, and Garlic: Cooking Secrets of the
     Mediterranean
Fifty Years in Buckingham Palace Kitchens
Sushi, Anyone?
```

式サブクエリでの *group by* と *having* の使用

修飾されていない比較演算子によって指定されるサブクエリは単一の値を返す必要があるので、group by 句や having 句が単一の値を返すとわかっていない限り、これらのサブクエリには group by 句や having 句を含めることはできません。

たとえば、次のクエリは、trad_cook カテゴリで最も安い価格の本よりも高い価格の本を検索します。

```
select title
from titles
where price >
```

```
(select min(price)
 from titles
 group by type
 having type = "trad_cook")
```

式サブクエリでの *distinct* の使用

修飾されていない比較演算子で指定されるサブクエリは、多くの場合、単一の値を返すために distinct キーワードを含めます。

たとえば、distinct がないと、次のサブクエリは複数の値を返すので失敗します。

```
select pub_name from publishers
    where pub_id =
        (select distinct pub_id
        from titles
        where pub_id = publishers.pub_id)
```

限定述語サブクエリ

限定述語サブクエリは、0 個以上の値のリストを返すものであり、any、all、in、または exists によって接続される where または having 句にあるサブクエリです。any または all のサブクエリ演算子は比較演算子を修飾します。

限定述語サブクエリには 3 つのタイプがあります。

- any/all サブクエリ。修飾された比較演算子で指定されるサブクエリ、これは、group by または having 句を含む可能性があり、次のような一般形式を使用する。

  [select、insert、update、delete 文またはサブクエリの始まり]

  ```
  where expression comparison_operator [any | all]
      (subquery)
  ```

  [select、insert、update、delete 文またはサブクエリの終わり]

- in/not in サブクエリ。in または not in で指定されたサブクエリは次の一般形式を使用する。

  [select、insert、update、delete 文またはサブクエリの始まり]

  ```
  where expression [not] in (subquery)
  ```

  [select、insert、update、delete 文またはサブクエリの終わり]

- exists/not exists サブクエリ。exists または not exists によって指定されたサブクエリは、次の一般形式を使用する存在確認テストである。

  [select、insert、update、delete 文またはサブクエリの始まり]

  ```
  where [not] exists (subquery)
  ```

[select、insert、update、delete 文またはサブクエリの終わり]

Adaptive Server は限定述語サブクエリでキーワード distinct を許可していますが、distinct が含まれていないかのようにサブクエリを処理します。

any と *all* のあるサブクエリ

キーワード all および any は、サブクエリを指定する比較演算子を修飾します。

any は、サブクエリで <、>、または = とともに使用されると、サブクエリで取り出された任意の値が外部文の where または having 句の値に一致するときに結果を返します。

all は、< または > とともにサブクエリで使用されると、サブクエリで取り出されたすべての値が外部文の where または having 句の値に一致するときに結果を返します。

any および all の構文は次のとおりです。

```
{where | having} [not]
    expression comparison_operator {any | all} (subquery)
```

> 比較演算子を使用する例には、次のものがあります。

- > all は、どの値よりも大きい、または、最大値よりも大きいことを意味する。たとえば、> all (1, 2, 3) は 3 よりも大きいという意味になる。
- >any は、少なくとも 1 つの値よりも大きい、または、最小値よりも大きいことを意味する。このため、> any (1, 2, 3) は 1 よりも大きいという意味になる。

all を持つサブクエリを指定して比較演算子が値を何も返さない場合、クエリ全体が失敗します。

all および any の扱いには、注意が必要です。たとえば、「New Age Books 社で出版されたどのような本よりも高い前払い金を要求した本はどれか」(Which books commanded an advance greater than any book published by New Age Books?) といった質問をするとします。

この質問を言い換えて SQL に「翻訳」しやすくすると、「New Age Books 社が支払った最高の前払い金よりも高い前払い金を要求した本はどれか」(“Which books commanded an advance greater than the largest advance paid by New Age Books?”) になります。ここで必要なのは、any キーワードではなく、all キーワードです。

```
select title
from titles
where advance > all
   (select advance
    from publishers, titles
    where titles.pub_id = publishers.pub_id
    and pub_name = "New Age Books")

title
```

```
---------------------------------------
The Gourmet Microwave
```

それぞれのタイトルに対して、外部クエリは、titles テーブルからタイトルと前払い金を取得して、これらをサブクエリから戻された New Age Books の前払い金と比較します。外部クエリはリストの中で最も大きな値を見て、対象のタイトルがより大きな値を要求したかどうかを判断します。

> *all* はすべての値よりも大きいことを意味する

> all 演算子は、外部クエリの条件をローが満たすために、サブクエリを指定するカラムにある値が、サブクエリによって戻されたそれぞれの値よりも大きい必要があるという意味です。

たとえば、mod_cook カテゴリの中で最も高価な本よりも高い価格の本を検索するには、次のようになります。

```
select title from titles where price > all
    (select price from titles
    where type = "mod_cook")

title
--------------------------------------------------
But Is It User Friendly?
Secrets of Silicon Valley
Computer Phobic and Non-Phobic Individuals:
    Behavior Variations
Onions, Leeks, and Garlic: Cooking Secrets of
    the Mediterranean

(4 rows affected)
```

ただし、内部クエリから返された集合が null を含んでいる場合、クエリは 0 個のローを返します。これは、null が「不定の値」を示すためであり、不定の値よりも比較している値が大きいかどうかを判別するのは不可能なためです。

たとえば、popular_comp カテゴリの中で最も高価な本よりも価格の高い本を検索しようとします。

```
select title from titles where price > all
    (select price from titles
    where type = "popular_comp")

title
--------------------------------------------------

(0 rows affected)
```

サブクエリは本の 1 つである『Net Etiquette』の価格が null であることを検出するので、返されるローはありません。

= *all* はすべての値に対して等価であることを意味する

= all 演算子は、ローが外部クエリを満たすために、サブクエリを指定するカラムにある値が、サブクエリによって返される値のリストにあるそれぞれの値と同じである必要があることを意味しています。

たとえば、次のクエリは、郵便番号で、同じ都市に住んでいる作家を識別します。

```
select au_fname, au_lname, city
from authors
where city = all
     (select city
     from authors
     where postalcode like "946%")
```

> *any* は少なくともある 1 つの値よりは大きいことを意味する

> any は、ローが外部クエリを満たすために、サブクエリを指定するカラムにある値が、サブクエリによって返されるリストにある少なくとも 1 つの値よりも大きい必要があることを意味しています。

次に示すのは、any によって修飾された比較演算子の例です。これは、New Age Books によって支払われた前払い金よりも高い前払い金がある本をそれぞれ検索します。

```
select title
from titles
where advance > any
   (select advance
    from titles, publishers
    where titles.pub_id = publishers.pub_id
    and pub_name = "New Age Books")

title
--------------------------------------------------
The Busy Executive's Database Guide
Cooking with Computers: Surreptitious Balance
    Sheets
You Can Combat Computer Stress!
Straight Talk About Computers
The Gourmet Microwave
But Is It User Friendly?
Secrets of Silicon Valley
Computer Phobic and Non-Phobic Individuals:
    Behavior Variations
Is Anger the Enemy?
Life Without Fear
Emotional Security: A New Algorithm
Onions, Leeks, and Garlic: Cooking Secrets of
    the Mediterranean
Fifty Years in Buckingham Palace Kitchens
Sushi, Anyone?
```

外部クエリによって選択されたそれぞれの本に対して、内部クエリは New Age Books によって支払われた前払い金のリストを検索します。外部クエリは、このリスト内のすべての値を見て、対象のタイトルがそれらの値のどれよりも高い前払い金を要求しているかどうかを判断します。つまり、この例は New Age Books によって支払われる最低額以上の前払い金のある本を検索します。

サブクエリが何も値を返さない場合、クエリ全体が失敗します。

= *any* は何らかの値と同じ意味になる

= any 演算子は存在するかどうかのチェックを行うものであり、in と等価です。たとえば、出版社の所在地と同じ都市に住んでいる作家を検索するには、=any または in のいずれかを使用します。

```
select au_lname, au_fname
from authors
where city = any
   (select city
    from publishers)
select au_lname, au_fname
from authors
where city in
   (select city
    from publishers)

au_lname       au_fname
-------------- --------------
Carson         Cheryl
Bennet         Abraham
```

ただし、!= any 演算子は not in とは異なります。!= any 演算子は「not = a または not = b または not = c」という意味であり、not in は「not = a かつ not = b かつ not = c」という意味です。

たとえば、出版社がまったくない都市に住んでいる作家を検索するために、次のように指定するとします。

```
select au_lname, au_fname
from authors
where city != any
   (select city
    from publishers)
```

その結果には 23 人すべての作家が含まれます。これは、どの作家も出版社がないどこかの都市に住んでおり、かつ 1 つの都市だけに住んでいるためです。

まず、内部クエリによって出版社のあるすべての都市が検索され、次に、外部クエリによって、それぞれの都市に対して、そこに住んでいない作家が検索されます。

次に、同じクエリを not in で置き換えた場合に、どうなるかを示します。

```
select au_lname, au_fname
```

```
from authors
where city not in
   (select city
    from publishers)
```

```
au_lname                au_fname
--------------          ------------
White                   Johnson
Green                   Marjorie
O'Leary                 Michael
Straight                Dick
Smith                   Meander
Dull                    Ann
Gringlesby              Burt
Locksley                Chastity
Greene                  Morningstar
Blotchet-Halls          Reginald
Yokomoto                Akiko
del Castillo            Innes
DeFrance                Michel
Stringer                Dirk
MacFeather              Stearns
Karsen                  Livia
Panteley                Sylvia
Hunter                  Sheryl
McBadden                Heather
Ringer                  Anne
Ringer                  Albert
```

これが求めていた結果です。その中には、Cheryl Carson と Abraham Bennet 以外のすべての作家が含まれており、この 2 人は Berkeley に住んでいて、そこには Algodata Infosystems があります。

not in と同じ意味を持つ !=all を使用すると、同じ結果が得られます。

```
select au_lname, au_fname
from authors
where city != all
   (select city
    from publishers)
```

in を使用したサブクエリ

キーワード in で指定されたサブクエリは 0 以上の結果のリストを返します。たとえば、次のクエリはビジネス書を出版した出版社の名前を検索します。

```
select pub_name
from publishers
where pub_id in
   (select pub_id
    from titles
```

```
    where type = "business")

pub_name
---------------------------------------
New Age Books
Algodata Infosystems
```

この文は 2 つの手順で評価されます。まず、内部クエリが、ビジネス書を発行した出版社の ID 番号、1389 と 0736 を返します。次に、これらの値は外部クエリに代入され、外部クエリが publishers テーブルを検索し、ID 番号とともにある名前を検出します。クエリは次のようになります。

```
select pub_name
from publishers
where pub_id in ("1389", "0736")
```

サブクエリを使ってこのクエリを作成する別の方法として、次のものがあります。

```
select pub_name
from publishers
where "business" in
   (select type
    from titles
    where pub_id = publishers.pub_id)
```

外部クエリの where キーワードに続く式は、カラム名だけでなく定数にもできます。定数とカラム名の組み合わせなど、その他の種類の式を使うこともできます。

前述のクエリは、他の多くのサブクエリと同様、代替的にジョイン・クエリとして編成することもできます。

```
select distinct pub_name
from publishers, titles
where publishers.pub_id = titles.pub_id
and type = "business"
```

このクエリと先ほどのサブクエリはどれも、ビジネス書を出版した出版社を検索します。重複を削除するために distinct キーワードを使うことが必要な場合もありますが、すべては同じだけ正確であり、同じ結果を生成します。

ただし、サブクエリよりもジョイン・クエリを使用する長所の 1 つは、ジョイン・クエリが複数のテーブルのカラムを結果に示すことです。たとえば、結果にビジネス書のタイトルを含めるには、次のようにジョインの方を使用する必要があります。

```
select pub_name, title
from publishers, titles
where publishers.pub_id = titles.pub_id
and type = "business"
```

```
pub_name             title
--------------------  ----------------------------------------
Algodata Infosystems  The Busy Executive's Database Guide
```

```
Algodata Infosystems    Cooking with Computers: Surreptitious
                        Balance Sheets
New Age Books           You Can Combat Computer Stress!
Algodata Infosystems    Straight Talk About Computers
```

次に、サブクエリまたはジョイン・クエリを使用する別の例、「カリフォルニアに住んでいて、その本の印税が 30% よりも少ない 2 番めの共著者をすべて検索する」ための文を示します。サブクエリを使うと、文は次のようになります。

```
select au_lname, au_fname
from authors
where state = "CA"
and au_id in
   (select au_id
    from titleauthor
    where royaltyper < 30
    and au_ord = 2)

au_lname                 au_fname
------------------------ ------------
MacFeather               Stearns
```

外部クエリはカリフォルニアに住んでいる 15 人の作家のリストを生成します。次に内部クエリが評価され、条件を満たす作家の ID リストを生成します。

内部クエリと外部クエリの両方で where 句に複数の条件を含めることができる点に注意してください。

ジョインを使うと、クエリは次のように表されます。

```
select au_lname, au_fname
from authors, titleauthor
where state = "CA"
and authors.au_id = titleauthor.au_id
  and royaltyper < 30
  and au_ord = 2
```

ジョインは常にサブクエリとして表すことができます。サブクエリは多くの場合、ジョインとして表すことができます。

not in を使用したサブクエリ

キーワード句 not in で指定されたサブクエリも、0 個以上の値のリストを返します。not in は、「not = a かつ not = b かつ not = c」という意味です。

このクエリは、「in を使用したサブクエリ」(168 ページ)にある例の逆で、ビジネス書を発行していない出版社の名前を検索します。

```
select pub_name from publishers
where pub_id not in
   (select pub_id
    from titles
```

```
    where type = "business")

pub_name
----------------------------------------
Binnet & Hardley
```

not in が in の代わりに使用されている点を除けば、このクエリは前のクエリと同じです。ただし、この文をジョインに変えることはできません。「不等価」ジョインは、ビジネス書ではない何冊かの本を出している出版社を検索します。等号を基にしていないジョインの意味の解釈については、「第 4 章 ジョイン：複数テーブルからのデータの検索」を参照してください。

null とともに *not in* を使用したサブクエリ

not in を使ったサブクエリは、外部クエリのそれぞれのローに対して値の集合を返します。外部クエリの値が内部クエリによって返された集合にない場合、not in は TRUE と評価し、また、外部クエリは対象のレコードを結果に表示します。

ただし、内部クエリによって返された集合が一致する値を含まず、null を含む場合、not in は UNKNOWN を返します。これは、null が「不定の値」を意味するためであり、検索する値が不定の値を含む集合の中にあるかどうかを判別することは不可能であるためです。外部クエリはこのローを削除します。次に例を示します。

```
select pub_name
    from publishers
    where $100.00 not in
        (select price
         from titles
         where titles.pub_id = publishers.pub_id)

pub_name
------
New Age Books
```

New Age Books は、価格が 100 ドルになる本は出版しない唯一の出版社です。Binnet & Handley と Algodata Infosystems は、両方とも価格が未決定の本を出版するので、検索結果には含まれません。

exists を使用したサブクエリ

サブクエリに exists キーワードを使用すると、そのサブクエリからの何らかの結果が存在するかどうかをテストできます。

{where | having} [not] exists (*subquery*)

つまり、外部クエリの where 句は、サブクエリによって返されたローが存在するかどうかをテストします。サブクエリは実際には何もデータを生成しませんが、TRUE または FALSE の値を返します。

たとえば、次のクエリはビジネス書を出版したすべての出版社の名前を検索します。

```
select pub_name
from publishers
where exists
     (select *
    from titles
    where pub_id = publishers.pub_id
    and type = "business")

pub_name
----------------------------------------
New Age Books
Algodata Infosystems
```

このクエリの結果を概念的に説明するために、それぞれの出版社の名前を順番に考えます。この値が少なくとも 1 つのローを返す原因になるのでしょうか。言い換えると、この値が存在のテストを TRUE に評価する原因になるのでしょうか。

前述のクエリの結果、2 番目の出版社の名前は Algodata Infosystems であり、この会社の ID 番号は 1389 です。pub_id が 1389 であり、type が business であるローが titles テーブルにあるでしょうか。あるのであれば、“Algodata Infosystems” は選択されている値に含まれるはずです。同じ処理が、その他の出版社の名前それぞれに対して繰り返されます。

exists のあるサブクエリは、次の点で他のサブクエリとは異なります。

- キーワード exists の前には、カラム名、定数、または、その他の式は指定されない。
- サブクエリ exists は、データを返すのではなく、TRUE または FALSE を評価する。
- サブクエリの select リストは、通常、アスタリスク (*) で構成される。サブクエリで指定された条件を満たすローが存在するかしないかをテストしているだけなので、カラム名を指定する必要はない。それ以外は、exists のあるサブクエリの select リスト規則は、標準の select リストの規則と同じである。

exists キーワードは、サブクエリ以外の代替表現がないことが多いので、非常に重要になります。実際、exists によって指定されたサブクエリは、常に相関サブクエリです (「相関サブクエリの使用」(176 ページ) を参照してください)。

exists のあるいくつかのクエリは別の方法で表すことができませんが、in を使用したり、any または all によって修飾された比較演算子を使用したりするすべてのクエリは exists で表すことができます。exists を使用する文と、それと等価な代替表現を使っている文のいくつかの例を次に示します。

次に、出版社と同じ都市に住んでいる作家を検索する方法を 2 つ示します。

```
select au_lname, au_fname
from authors
```

```
where city = any
   (select city
    from publishers)

select au_lname, au_fname
from authors
where exists
     (select *
    from publishers
    where authors.city = publishers.city)

au_lname          au_fname
--------------    --------------
Carson            Cheryl
Bennet            Abraham
```

次に示すのは、文字 “B” で始まる都市にある出版社によって発行された本のタイトルを検索する 2 つのクエリです。

```
select title
from titles
where exists
     (select *
    from publishers
    where pub_id = titles.pub_id
    and city like "B%")

select title
from titles
where pub_id in
   (select pub_id
    from publishers
    where city like "B%")

title
--------------------------------------------------
You Can Combat Computer Stress!
Is Anger the Enemy?
Life Without Fear
Prolonged Data Deprivation: Four Case Studies
Emotional Security: A New Algorithm
The Busy Executive’s Database Guide
Cooking with Computers: Surreptitious Balance
    Sheets
Straight Talk About Computers
But Is It User Friendly?
Secrets of Silicon Valley
Net Etiquette
```

not exists を使用したサブクエリ

not exists は、not exists を使用しているサブクエリがローを返さない場合に where 句が満たされる点以外は、exists と同じです。

たとえば、ビジネス書を出版していない出版社の名前を検索するには、クエリは次のようになります。

```
select pub_name
from publishers
where not exists
     (select *
    from titles
    where pub_id = publishers.pub_id
    and type = "business")
pub_name
----------------------------------------
Binnet & Hardley
```

次のクエリは売上がない本のタイトルを検索します。

```
select title
from titles
where not exists
   (select title_id
    from salesdetail
    where title_id = titles.title_id)

title
----------------------------------------
The Psychology of Computer Cooking
Net Etiquette
```

exists を使用した積と差の検索

exists および not exists を使用するサブクエリにより、積および差の集合論的演算を実行できます。2 つの集合の積は、オリジナルの両方の集合に属しているすべての要素を含んでいます。また、差は、最初の集合だけに属している要素を含んでいます。

city カラムでの authors と publishers の積は、作家が住んでおり、かつ出版社がある都市の集合です。

```
select distinct city
from authors
where exists
  (select *
   from publishers
   where authors.city = publishers.city)

city
--------------------
```

```
Berkeley
```

city カラムでの authors と publishers の差は、作家は住んでいても出版社のない都市の集合、つまり、Berkeley 以外のすべての都市です。

```
select distinct city
from authors
where not exists
  (select *
   from publishers
   where authors.city = publishers.city)

city
--------------------
Gary
Covelo
Oakland
Lawrence
San Jose
Ann Arbor
Corvallis
Nashville
Palo Alto
Rockville
Vacaville
Menlo Park
Walnut Creek
San Francisco
Salt Lake City
```

SQL 抽出テーブルを使用したサブクエリ

SQL 抽出テーブルは、サブクエリの from 句で使用できます。たとえば、次のクエリはビジネス書を出版した出版社の名前を検索します。

```
select pub_name from publishers
   where "business" in
      (select type from
         (select type from titles, publishers
             where titles.pub_id = publishers.pub_id)
       dt_titles)
```

ここで、dt_titles は最も内側の select 文で定義された SQL 抽出テーブルです。

SQL 抽出テーブルは、サブクエリの使用が有効な場所であればどこでも、サブクエリの from 句で使用できます。「第 9 章 SQL 抽出テーブル」を参照してください。

相関サブクエリの使用

これまで示したクエリの多くは、サブクエリを一度実行し、その結果を外部クエリの where 句に代入することによって評価します。つまり、これらは非相関サブクエリです。繰り返しサブクエリまたは「相関サブクエリ」を含むクエリでは、サブクエリの値は外部クエリに依存します。サブクエリは、外部クエリによって選択されるそれぞれのローに対して 1 回ずつ、繰り返し実行されます。

次の例は、本の 100% の印税を得るすべての作家の名前を検索します。

```
select au_lname, au_fname
from authors
where 100 in
   (select royaltyper
    from titleauthor
    where au_id = authors.au_id)

au_lname        au_fname
--------------  ----------
White           Johnson
Green           Marjorie
Carson          Cheryl
Straight        Dick
Locksley        Chastity
Blotchet-Hall   Reginald
del Castillo    Innes
Panteley        Sylvia
Ringer          Albert

(9 rows affected)
```

これまでのほとんどの例とは異なり、この文にあるサブクエリはメイン・クエリと別に処理することはできません。このサブクエリは、authors.au_id の値が必要ですが、この値は変数なので、Adaptive Server が authors テーブルの異なるローを調べるに従って値が変わります。

前述のクエリは、次のように評価されます。Transact-SQL は、この変数の値を内部クエリのローに代入することによって、authors テーブルのそれぞれのローについて結果に取り込むかどうかを調べます。たとえば、Transact-SQL が Johnson White のローを最初に調査するとします。Transact-SQL は、内部クエリに “72-32-1176” を代入します。この結果、authors.au_id の値は “72-32-1176” になります。

```
select royaltyper
from titleauthor
where au_id = "172-32-1176"
```

結果は 100 であり、外部クエリは次のように評価します。

```
select au_lname, au_fname
from authors
where 100 in (100)
```

where 条件が true なので、Johnson White のローが結果に含まれます。Abraham Bennet のローで同じ手順を実行すると、ローが結果に含まれない過程を確認することができます。

次のクエリは、Transact-SQL 外部ジョインの外部メンバとして相関変数を使用します。

```
select t2.b1, (select t2.b2 from t1 where t2.b1 *= t1.a1) from t2
```

相関名のある相関サブクエリ

相関サブクエリを使うと、複数の出版社が出版した本の種類を検索することができます。

```
select distinct t1.type
from titles t1
where t1.type in
   (select t2.type
    from titles t2
    where t1.pub_id != t2.pub_id)

type
--------------------
business
psychology
```

相関名は、titles テーブルが 2 つの異なる役割を果たすため、その役割を区別するために次のクエリで必要になります。このネストしたクエリはセルフジョイン・クエリと同等です。

```
select distinct t1.type
from titles t1, titles t2
where t1.type = t2.type
and t1.pub_id != t2.pub_id
```

比較演算子のある相関サブクエリ

式サブクエリは相関サブクエリにすることができます。たとえば、心理学の本の中で、平均よりも販売数が少ない心理学の本の販売を検索するには次のようになります。

```
select s1.ord_num, s1.title_id, s1.qty
from salesdetail s1
where title_id like "PS%"
and s1.qty <
   (select avg(s2.qty)
    from salesdetail s2
    where s2.title_id = s1.title_id)

ord_num            title_id     qty
------------------ --------     ---
```

```
91-A-7              PS3333        90
91-A-7              PS2106        30
55-V-7              PS2106        31
AX-532-FED-452-2Z7 PS7777       125
BA71224             PS7777       200
NB-3.142            PS2091       200
NB-3.142            PS7777       250
NB-3.142            PS3333       345
ZD-123-DFG-752-9G8 PS3333       750
91-A-7              PS7777       180
356921              PS3333      200
```

外部クエリは sales テーブル (または s1) のローを 1 つずつ選択します。サブクエリは、外部クエリでの選択の対象になるそれぞれの販売に対して平均の量を計算します。s1 の可能な値それぞれに対して、Transact-SQL はサブクエリを評価して、量が計算された平均よりも少ない場合は、対象のレコードを結果に含めます。

場合によっては、相関サブクエリは group by 文のようになります。同じ種類の本の平均よりも高価な価格の本のタイトルを検索するには、クエリは次のようになります。

```
select t1.type, t1.title
from titles t1
where t1.price >
   (select avg(t2.price)
    from titles t2
    where t1.type = t2.type)
```

```
type         title
---------    ---------------------------------------
business     The Busy Executive's Database Guide
business     Straight Talk About Computers
mod_cook     Silicon Valley Gastronomic Treats
popular_comp But Is It User Friendly?
psychology   Computer Phobic and Non-Phobic
             Individuals: Behavior Variations
psychology   Prolonged Data Deprivation: Four Case
             Studies
trad_cook    Onions, Leeks, and Garlic: Cooking
             Secrets of the Mediterranean
```

t1 の取り得るそれぞれの値に対して、Transact-SQL はサブクエリを評価し、ローの価格値が計算された値よりも大きな場合、そのローを結果に含めます。平均価格が計算されるローはサブクエリの where 句で制限されるので、種類を使用して明示的にグループ化する必要はありません。

having 句での相関サブクエリ

限定述語サブクエリは相関サブクエリにすることができます。

次の例では、外部クエリの having 句にある相関サブクエリは、前渡し金額の最大値が、指定のグループ内での平均前渡し金額の 2 倍よりも大きい本の種類を検索しています。

```
select t1.type
from titles t1
group by t1.type
having max(t1.advance) >= any
   (select 2 * avg(t2.advance)
   from titles t2
   where t1.type = t2.type)

type
----------
mod_cook
```

前述のサブクエリは、外部クエリで定義されたそれぞれのグループに対して 1 回、つまり、本のそれぞれの種類に対して 1 回ずつ評価されます。

第 6 章

データ型の使用と作成

「データ型」は、テーブルの各カラムが保持する情報の種類と、その情報がどのように格納されるかを定義します。カラムを定義するときに Adaptive Server システムのデータ型を使用できますが、ユーザ定義データ型を作成することもできます。

Transact-SQL データ型の概要

Transact-SQL では、データ型は情報のタイプ、サイズ、およびテーブル・カラム、ストアド・プロシージャ・パラメータ、ローカル変数の記憶フォーマットを指定します。たとえば int (integer) データ型は、プラスまたはマイナス 2^{31} の範囲の整数を格納し、tinyint (tiny integer) データ型は 0 から 255 の整数だけを格納します。

Adaptive Server は、複数のシステム・データ型、および 2 つのユーザ定義データ型、timestamp と sysname を提供します。システム・データ型に基づいてユーザ定義データ型を構築するには、sp_addtype を使用します。

カラム、ローカル変数、またはパラメータを宣言するときに、システム・データ型かユーザ定義データ型を指定する必要があります。次の例は、create table 文内でシステム・データ型 char、numeric、および money を使用して、カラムを定義します。

```
create table sales_daily
    (stor_id char(4),
    ord_num numeric(10,0),
    ord_amt money)
```

次の例は、declare 文内でシステム・データ型 bit を使用して、ローカル変数を定義します。

```
declare @switch bit
```

この章で説明するデータ型を使用してカラム、ローカル変数、およびパラメータを宣言する方法については、以降の章でさらに詳しく説明します。sp_help を使用して、既存のテーブルのカラムにどのデータ型が定義されているかを確認できます。

システム提供のデータ型の使用

さまざまなタイプの情報に合わせて提供されるシステム提供のデータ型と Adaptive Server が認識する同義語、およびそれぞれのデータ型の範囲と記憶サイズを、表 6-1 に示します。システム・データ型は、Adaptive Server では大文字でも小文字でも入力できますが、小文字で出力されます。Adaptive Server 提供のデータ型のほとんどは予約語ではなく、他のオブジェクトの名前に使用できます。

表 6-1: Adaptive Server のシステム・データ型

データ型 (種類別)	同義語	範囲	記憶サイズ (バイト数)
真数値：整数			
bigint		2^{63} と $-2^{63} - 1$ (-9,223,372,036,854,775,808 ～ +9,223,372,036,854,775,807) の間の整数値	8
int	integer	2^{31} -1 (2,147,483,647) ～ -2^{31} (-2,147,483,648)	4
smallint		2^{15} -1 (32,767) ～ -2^{15} (-32,768)	2
tinyint		0 ～ 255 (負の数は使用できない)	1
unsigned bigint		0 ～ 18,446,744,073,709,551,615 の間の整数値	8
unsigned int		0 ～ 4,294,967,295 の間の整数値	4
unsigned smallint		0 ～ 65535 の間の整数値	2
真数値：小数			
数値 (precision、scale)		10^{38} -1 ～ -10^{38}	17 ～ 2
decimal (precision、scale)	dec	10^{38} -1 ～ -10^{38}	17 ～ 2
概数値			
float (precision)		マシンに依存する	デフォルト精度が 16 未満の場合は 4、 デフォルト精度が 16 以上の場合は 8。
double precision		マシンに依存する	8
real		マシンに依存する	4
通貨			
smallmoney		-214,748.3647 ～ 214,748.3648	4

データ型 (種類別)	同義語	範囲	記憶サイズ (バイト数)
money		922,337,203,685,477.5807 ～ -922,337,203,685,477.5808	8
日／時			
smalldatetime		1900 年 1 月 1 日 ～ 2079 年 6 月 6 日	4
datetime		1753 年 1 月 1 日 ～ 9999 年 12 月 31 日	8
date		0001 年 1 月 1 日 ～ 9999 年 12 月 31 日	4
time		12:00:00 AM から 11:59:59:999 PM まで	4
bigdatetime		0001 年 1 月 1 日 ～ 9999 年 12 月 31 日 および 12:00:00.000000 AM ～ 11:59:59.999999 PM	8
bigtime		12:00:00.000000 AM から 11:59:59:999999 PM まで	8
文字			
char(n)	character	ページ・サイズ	n
varchar(n)	character varying、char varying	ページ・サイズ	実際のエントリの長さ
unichar	Unicode 文字	ページ・サイズ	*n* * *@@unicharsize* (*@@unicharsize* は 2)
univarchar	Unicode 文字 varying、char varying	ページ・サイズ	実際の文字数 * *@@unicharsize*
nchar(n)	national character、national char	ページ・サイズ	*n* * *@@ncharsize*
nvarchar(n)	nchar varying、national char varying、national character varying	ページ・サイズ	*@@ncharsize* * 文字数
text		2^{31} -1 (2,147,483,647) バイト以下	初期化前は 0、初期化後は 2K の倍数
unitext		1,073,741,823 以下の Unicode 文字	初期化前は 0、初期化後は 2K の倍数
バイナリ			
binary(n)		ページ・サイズ	*n*
varbinary(n)		ページ・サイズ	実際のエントリの長さ
image		2^{31} -1 (2,147,483,647) バイト以下	初期化前は 0、初期化後は 2K の倍数
ビット			
bit		0 または 1	(1 バイトで 8 つまでの bit カラムを保持する)

真数値型：整数

Adaptive Server は、整数を格納するためのデータ型として、bigint、int、smallint、tinyint、unsigned bigint、unsigned int、unsigned smallint を提供します。これらのデータ型は真数値型で、算術演算の間、その正確性を保ちます。

格納する数字の予想サイズに基づいて、整数型を選択します。内部記憶サイズはデータ型ごとに異なります。

任意の整数型から異なる整数型への暗黙的変換は、値が変換先の型の範囲内にある場合のみサポートされます。

符号なしの整数データ型を使用すると、必要な記憶域のサイズを増やすことなく、既存の integer データ型に対する正の値の範囲を拡大できます。つまり、これらのデータ型の符号付きバージョンの範囲は、正の方向と負の方向の両方に広がります (たとえば、-32 ～ +32)。一方、符号なしバージョンの範囲は、正の方向のみに広がります。表 6-2 は、これらのデータ型の符号付きバージョンと符号なしバージョンの範囲の説明です。

表 6-2: 符号付きデータ型と符号なしデータ型の範囲

データ型	符号付きデータ型の範囲	データ型	符号なしデータ型の範囲
bigint	-2^{63} ～ 2^{63} - 1 (-9,223,372,036,854,775,808 ～ +9,223,372,036,854,775,807) の間の整数値	unsigned bigint	0 ～ 18,446,744,073,709,551,615 の間の整数値
int	-2^{31} ～ 2^{31} - 1 (2,147,483,648 ～ -2,147,483,647) の間の整数値	unsigned int	0 ～ 4,294,967,295 の間の整数値
smallint	-2^{15} ～ 2^{15} - 1 (-32,768 ～ 32,767) の間の整数値	unsigned smallint	0 ～ 65535 の間の整数値

真数値型：小数点数

小数点を含む数字には、真数値型 numeric および decimal を使用します。numeric および decimal カラムに格納されるデータは、ディスク領域を節約するためにパックされ、算術演算後、その正確性を最下位有効桁数まで保ちます。numeric と decimal の 2 つのデータ型は、identity カラムには位取り 0 の numeric 型だけが使用できるという点を除いて、同一です。

真数値型には、2 つのオプションのパラメータ、precision と scale を、カッコで囲み、カンマで区切って指定できます。

datatype [(*precision* [, *scale*])]

Adaptive Server は、「**精度**」と「**位取り**」の各組み合わせを、個別のデータ型として定義します。たとえば numeric(10,0) と numeric(5,0) は、別々の 2 つのデータ型です。精度と位取りは、decimal または numeric カラムに格納できる値の範囲を決定します。

- precision は、カラムに格納できる 10 進の桁の最大数を指定します。小数点の左または右のすべての桁がこれに含まれます。1 から 38 桁までの範囲の精度を指定できます。または 18 桁のデフォルトの精度を使用できます。

- 位取りは、小数点の右側に格納できる最大桁数を指定します。scale　は、precision 以下でなければなりません。1 から 38 桁までの範囲の位取りを指定できます。または 0 桁のデフォルトの位取りを使用できます。

位取りが 0 の真数値型は、小数点なしで表示されます。カラムの精度または位取りのいずれかを超える値は入力できません。

numeric または decimal カラムの記憶サイズは、その精度によって異なります。1 桁または 2 桁のカラムの場合、格納領域は 2 バイト必要です。記憶サイズは、精度が 2 桁追加されるごとに 1 バイトずつ、最大 17 バイトまで増加します。

概数値データ型

数値型 float、double precision、real は、算術演算中の丸めを許可します。

概数値データ型は、データをバイナリの形式で補完するため、実数と比べてわずかに差が生じます。概数値が表示、出力、ホスト間で転送、または計算に使用されるときは、数字は常に精度を失います。isql は、小数点以下の有効桁数を 6 桁しか表示せず、残りを丸めます。『リファレンス・マニュアル：ビルディング・ブロック』の「第 1 章 システム・データ型とユーザ定義データ型」を参照してください。

概数値データ型を使用すると、広範囲にわたる値が保管できます。概数値型は、すべての集合関数とすべての算術演算をサポートします。

real および double precision 型は、オペレーティング・システムが提供する型を基に構築されます。float 型は、カッコで囲まれたオプションの精度を受け入れます。1 から 15 精度の float カラムは、real として格納されます。これより高い精度の場合は、double precision として格納されます。3 つの型のいずれの場合も、範囲および記憶精度はマシンによって異なります。

通貨データ型

money データ型 money および smallmoney は、通貨データを格納します。Adaptive Server はある通貨から他の通貨に変換する方法は提供しませんが、これらのデータ型は、米ドルおよびその他の 10 進法通貨に使用できます。money および smallmoney データには、modulo を除く算術演算と、すべての集合関数を使用できます。

money と smallmoney は、いずれも通貨単位の 10000 分の 1 まで正確ですが、表示のために、値を小数点以下 2 桁までに丸めます。デフォルトの出力フォーマットでは、3 桁ごとにカンマが挿入されます。

日付と時刻のデータ型

1753 年 1 月 1 日から 9999 年 12 月 31 日までの日付と時刻の情報を格納するには、datetime データ型と smalldatetime データ型を使用します。0001 年 1 月 1 日から 9999 年 12 月 31 日までの日付には date を使用し、12:00:00 AM から 11:59:59:999 には time を使用します。この範囲外の日付は、char または varchar 値として入力、格納、操作する必要があります。

- datetime カラムは、1753 年の 1 月 1 日から 9999 年 12 月 31 日までの日付を保持します。datetime 値は、プラットフォームの能力が対応可能であれば、300 分の 1 秒のレベルまで正確です。記憶サイズは 8 バイトです。基本の日付である 1900 年 1 月 1 日以降の日数に 4 バイト、1 日の時刻に 4 バイトを使用します。
- smalldatetime カラムは、1900 年 1 月 1 日から 2079 年 6 月 6 日までの日付を、分の単位まで正確に保持します。記憶サイズは 4 バイトです。1900 年 1 月 1 日以降の日数に 2 バイト、夜中の 12 時以降の分数に 2 バイトを使用します。
- bigdatetime カラムは、0001 年 1 月 1 日から 9999 年 12 月 31 日までの日付および 12:00:00.000000 AM から 11:59:59.999999 PM までの時刻を保持します。記憶サイズは 8 バイトです。bigdatetime 値はマイクロ秒まで正確です。bigdatetime の内部で使用される表現は、2000 年 1 月 1 日以降のマイクロ秒数を含む 64 ビットの整数です。
- bigtime カラムは、12:00:00.000000 AM から 11:59:59.999999 PM までの時刻を保持します。記憶サイズは 8 バイトです。bigtime 値はマイクロ秒まで正確です。bigtime の内部で使用される表現は、午前 0 時以降のマイクロ秒数を含む 64 ビットの整数です。
- date は、一重または二重引用符で囲まれた日付部分から構成されるリテラル値です。このカラムは、0001 年 1 月 1 日から 9999 年 12 月 31 日までの日付を保持できます。記憶サイズは 4 バイトです。
- time は、一重または二重引用符で囲まれた時刻部分から構成されるリテラル値です。このカラムは、12:00:00AM ～ 11:59:59:999PM の時刻を保持します。記憶サイズは 4 バイトです。

日付と時刻の情報は一重または二重の引用符で囲みます。大文字と小文字のどちらでも入力でき、日付部分の間にスペースを使用できます。Adaptive Server は、「第 7 章 データの追加、変更、転送、削除」で説明されているようなさまざまな日付エントリのフォーマットを認識します。ただし、0 または 00/00/00 のような値は拒否され、日付として認識されません。

日付のデフォルトの表示フォーマットは、“Apr 15 1987 10:23 p.m” です。convert 関数は他のフォーマットで使用できます。組み込み日付関数を使用して、datetime 値についていくつかの算術計算を実行することもできますが、time データ型を使用しないかぎり、Adaptive Server はミリ秒の値を丸めるか、またはトランケートする場合があります。

bigdatetime および bigtime の場合、表示される値はマイクロ秒の精度で示されます。bigdatetime と bigtime には、この増えた精度に対応するデフォルトの表示フォーマットがあります。

- hh:mi:ss.zzzzzzAM または PM
- hh:mi:ss.zzzzzz
- mon dd yyyy
  hh:mi:ss.zzzzzz
- yyyy-mm-dd
  hh:mi:ss.zzzzzz

文字データ型

英字、数字、および一重または二重の引用符で囲んで入力した記号で構成される文字列の格納には、文字データ型を使用します。like キーワードを使用してこれらの文字列から特定の文字を検索し、組み込み文字列関数を使用してそれらの内容を操作します。数字で構成される文字列は convert 関数で真数値および概数値データ型に変換され、算術に使用できます。

英語などのシングルバイト文字セットでは、char(*n*) データ型は固定長文字列を、varchar(*n*) データ型は可変長文字列を格納します。国別の文字でこれに相当するのが nchar(*n*) と nvarchar(*n*) で、日本語などのマルチバイト文字セットの固定長および可変長の文字列を格納します。unichar および univarchar データ型は固定長の Unicode 文字を格納します。n を使用して最大文字数を指定するか、またはある文字のデフォルトのカラム長を使用できます。ページ・サイズを超える文字列の場合は、text データ型を使用します。

表 6-3: 文字データ型

データ型	保管するデータ
char(n)	社会保障番号や郵便番号などの固定長データ
varchar(n)	名前など、長さが大きく異なる可能性が高いデータ
unichar	固定長 Unicode データ、char と比較可能
univarchar	長さが大きく異なる可能性が高い Unicode データ、varchar と比較可能
nchar(n)	マルチバイト文字セットの固定長データ
nvarchar(n)	マルチバイト文字セットの可変長データ
text	データ・ページのリンク・リスト上の、2,147,483,647 バイトまでの出力可能文字
unitext	データ・ページのリンク・リスト上の、1,073,741,823 文字までの Unicode 文字

string_rtruncation on を設定していないと、Adaptive Server は、警告やエラーを出すことなく、指定されたカラムの長さまでエントリをトランケートします。『リファレンス・マニュアル：コマンド』を参照してください。空文字列 “ ” や ‘ ’ は、null としてではなく、シングル・スペースとして格納されます。したがって、“abc” + “ ” + “def” は、“abc def” と等価ですが、“abcdef” とは等価ではありません。

固定長カラムと可変長カラムは、動作が若干異なります。

- 固定長カラムのデータは、カラム長までブランクが埋め込まれます。char データ型と unichar データ型の場合、記憶サイズは *n* バイトです (unichar = n*@@unicharsize)。ncharの場合は、平均国別文字長 (*@@ncharsize*) の *n* 倍です。null 入力可の char、unichar、または nchar カラムを作成すると、Adaptive Server はそれを varchar、univarchar、または nvarchar カラムに変換して、これらのデータ型の格納規則を使用します。char と nchar の変数およびパラメータには、これは当てはまりません。
- 可変長カラムのデータは、後続ブランクが取り除かれます。記憶サイズはデータの実際の長さになります。varchar または univarchar カラムの場合、これは文字数です。nvarchar カラムの場合は、文字数かける平均文字長になります。可変長文字データに必要な領域は固定長データよりも少ない場合がありますが、可変長データはアクセスが多少遅くなります。

unichar データ型

unichar データ型と univarchar データ型は、Adaptive Server において Unicode の UTF-16 コード化をサポートします。これらのデータ型は、char データ型と varchar データ型から独立していますが、動作をミラーリングします。

たとえば、char と varchar で機能する組み込み関数は、unichar と univarchar でも機能します。ただし、unichar と univarchar は UTF-16 文字だけを格納し、char および varchar とは違ってデフォルトの文字セット ID やデフォルトのソート順 ID との関連はありません。

unichar/univarchar 文字は、1 文字あたり 2 バイトの記憶領域が必要です。unichar/univarchar カラムの宣言は、16 ビットの Unicode 値です。次に、20 バイトの記憶領域を要する、Unicode 値 10 の unichar カラムを 1 つ持つテーブルを作成する場合の例を示します。

```
create table unitbl (unicol unichar(10))
```

unichar/univarchar カラムの長さは、データ・ページのサイズにより制限されます。これは、char/varchar カラムの長さの場合と同じです。

Unicode サロゲート・ペアは、16 ビット Unicode 値 2 個分 (つまり 4 バイト分) の記憶領域を使用します。Unicode サロゲート・ペア ([0x010000..0x10FFFF] の範囲内にある文字を表す 16 ビット値のペア) を格納するカラムを宣言する場合は、このことに注意してください。デフォルトでは、Adaptive Server はサロゲート・ペアを適切に処理し、ペアを分断しません。Unicode データのトランケートは、char と varchar データのトランケートと同じ方法で処理されます。

unichar 式は、char 式を使用するあらゆる場所 (比較演算子、ジョイン、サブクエリなども含む) で使用できます。ただし、unichar と char の両方による混合モードの式は unichar として実行されます。上記のような演算子に含めることができる Unicode 値の個数は、unichar 文字列の最大サイズまでに制限されます。

「正規化処理」によって、特定の抽象文字シーケンスに対する、データベース内での表現が 1 つだけになるように、Unicode データが修正されます (正規化の詳細については、『パフォーマンス＆チューニング・シリーズ：基本』の「基本の概要」を参照)。多くの場合、発音区別符号が後に付いた文字が、事前結合済みの形式に置き換えられます。これにより、パフォーマンスが大幅に向上します。デフォルトでは、サーバはすべての Unicode データを正規化するものとみなします。

関係式

unichar または univarchar の式を最低 1 つ含む関係式は、すべて Unicode のデフォルトのソート順に従います。一方の式が unichar で、他方の式が varchar (nvarchar、char、または nchar) である場合、後者が unichar に暗黙的に変換されます。

where 句では、表 6-4 に示す式が最も頻繁に使用されます。これらの句では、論理演算子と組み合わせることができます。

Unicode 文字データの比較では、「より小さい」は Unicode のデフォルトのソート順の先頭に近いことを、「より大きい」は末尾に近いことをそれぞれ意味します。「等しい」は、Unicode のデフォルトのソート順によって 2 つの値が区別されないことを意味します (ただし、これらは同じ値である必要はありません)。たとえば、事前結合済みの ê という文字は、文字 e とU+0302 から成る結合シーケンスと同じであるとみなす必要があります (事前結合済みの文字は、他の複数の文字の同等の文字列に細分化できる Unicode 文字です)。Unicode の正規化機能をオン (デフォルト) にしていれば、Unicode データは自動的に正規化され、正規化されていないデータがサーバに発生することはありません。

表 6-4: 関係式

expr1 op_compare [any | all] (subquery)	any または all を比較演算子と併用し、expr2 をサブクエリとすると、min または max が暗黙的に呼び出されます。たとえば、“expr1> any expr2” は、実際には “expr1> min(expr2)” と同じ意味になります。
expr1 [not] in (expression list) expr1 [not] in (subquery)	in 演算子は、expr2 の各要素が等しいかどうかをチェックします。これらの要素は、定数のリストである場合や、サブクエリの結果である場合があります。
expr1 [not] between expr2 and expr3	between 演算子は、範囲を指定します。この演算子は、実際には、“expr1 = expr2 and expr1<= expr3” を簡略化したものです。
expr1 [not] like "match_string" [escape"esc_char"]	like 演算子は、一致するパターンを指定します。Unicode データとのパターン一致検査のためのセマンティックは、通常の文字データと同じです。expr1 が unichar カラム名である場合、“match_string” は unichar 文字列と varchar 文字列のどちらにもすることができます。後者の場合は、varchar と unichar の間の変換が暗黙的に行われます。

ジョイン演算子

ジョイン演算子は、比較演算子と同じような形式で使用されます。実際、ジョインにはどの比較演算子も使用できます。unichar 型の式を最低 1 つ含む式は、Unicode のデフォルトのソート順に従います。一方の式の型が unichar で、他方の式の型が varchar (nvarchar、char、または nchar) である場合、後者が暗黙的に unichar に変換されます。

union 演算子

union 演算子は、unichar データと varchar データを同じように操作します。個々のクエリの対応カラムは、暗黙的に unichar に変換可能でなければなりません。そうでない場合は、明示的に変換する必要があります。

句と変更子

group by 句や order by 句で unichar カラムと univarchar カラムが使用されている場合、Unicode のデフォルトのソート順に従って、等しいかどうかが判定されます。distinct 変更子を使用する場合も同じです。

text データ型

text データ型は、別々のデータ・ページのリンク・リスト上の、2,147,483,647 バイトまでの出力可能文字を格納します。各ページは、最大で 1800 バイトのデータを格納します。

記憶領域を節約するには、text カラムを null として定義します。text カラムを非 null の insert または update で初期化すると、Adaptive Server はテキスト・ポインタを割り当て、2K データページ全体を割り付けて値を保持します。

コンポーネント統合サービスで接続されているデータベースを使用している場合、text データ型の処理にはいくつか異なる点があります。『コンポーネント統合サービス・ユーザーズ・ガイド』を参照してください。

text データ型の詳細については、「text データ、unitext データ、image データの変更」(235 ページ) および『リファレンス・マニュアル：ビルディング・ブロック』を参照してください。

unitext データ型

可変長の unitext データ型は、Unicode 文字で最大 1,073,741,823 文字 (2,147,483,646 バイト) まで保持できます。unitext は、text データ型を使用できる場所であれば、同じセマンティックで使用できます。unitext カラムは、Adaptive Server のデフォルト文字セットとは関係なく、UTF-16 コード化で保管されます。

unitext データ型は、text と同じ記憶メカニズムを使用します。記憶領域を節約するには、unitext カラムを null として定義します。unitext カラムを非 null の insert 句または update 句で初期化すると、Adaptive Server はテキスト・ポインタを割り当て、2K データ・ページ全体を割り付けて値を保持します。

unitext の利点は次のとおりです。

- Unicode 文字の大きなデータ。unichar データ型および univarchar データ型と併せて、Adaptive Server では Unicode データ型が完全にサポートされるため、多言語アプリケーションをインクリメンタル開発する場合に最適です。
- unitext は UTF-16 でデータを格納します。これは、Windows 環境と Java 環境で使用するネイティブなコードです。

「text データ、unitext データ、image データの変更」(235 ページ) および『リファレンス・マニュアル：ビルディング・ブロック』を参照してください。

バイナリ・データ型

バイナリ・データ型は、写真などの生のバイナリ・データを、16 進に似た表記で格納します。バイナリ・データは “0x” という文字で開始し、数字と A から F の大文字と小文字の任意の組み合わせを含みます。binary データと varbinary データ内で、“0x” の後に続く 2 桁は、数値の種類を示します。“00” は正の数値を表し、“01” は負の数値を表します。

入力値に “0x” がない場合、Adaptive Server はその値が ASCII 値であると想定して、値を変換します。

注意　Adaptive Server はバイナリ型をプラットフォーム固有の方法で操作します。実際の 16 進データには、hextoint および inttohex 関数を使用します。「第 16 章 クエリでの Transact-SQL 関数の使用」を参照してください。

長さが 255 バイトまでのデータを格納するには、binary(n) および varbinary(n) データ型を使用します。記憶領域の各バイトには、2 桁の 2 進数が保持されます。*n* でカラム長を指定するか、またはデフォルトの長さの 1 バイトを使用します。*n* より長い値を入力すると、Adaptive Server は、警告やエラーを発生することなく、エントリを指定の長さにトランケートします。

- すべてのエントリが同じような長さになることが予想されるデータには、固定長バイナリ型の binary(n) を使用します。binary カラムのエントリは、カラム長まで 0 が埋め込まれるので varbinary よりも多くの記憶領域が必要になる場合がありますが、アクセスは多少早くなります。
- 長さが大きく異なることが予想されるデータには、可変長バイナリ型の varbinary(n) を使用します。記憶サイズは、カラム長ではなく、入力したデータ値の実際のサイズになります。後続の 0 はトランケートされます。

null 入力可の binary カラムを作成すると、Adaptive Server はそれを varbinary カラムに変換して、そのデータ型の格納規則を使用します。

like キーワードでバイナリ文字列を検索して、組み込み文字列関数を使用して操作できます。

注意 特定の値を入力する正確な形式は使用しているハードウェアによって異なるので、バイナリ・データに関係する計算は、異なるプラットフォームで異なる結果を生成する場合があります。

---

image データ型

大きなブロックのバイナリ・データを外部データ・ページに格納するには、image データ型を使用します。image カラムは、テーブルの他のデータ記憶領域とは別のデータ・ページのリンク・リストに、2,147,483,647 バイトまでのデータを格納できます。

image カラムを非 null の insert または update で初期化すると、Adaptive Server はテキスト・ポインタを割り当て、2K データ・ページ全体を割り付けて値を保持します。各ページは、最大で 1800 バイトのデータを格納します。

記憶領域を節約するには、image カラムを null として定義します。トランザクション・ログに大きなブロックのバイナリ・データを保存せずに image データを追加するには、writetext を使用します。『リファレンス・マニュアル：コマンド』を参照してください。

image データ型は次の状況では使用できません。

- ストアド・プロシージャへのパラメータ (値をストアド・プロシージャのパラメータに渡す場合)、またはローカル変数。
- RPC (リモート・プロシージャ・コール) へのパラメータ。
- order by、compute、group by、または union 句内。
- インデックス内
- サブクエリまたはジョイン内
- where 句内。ただし、キーワード like で使用する場合を除く。
- + 連結演算子を指定した場合。
- トリガの if update 句内。

コンポーネント統合サービスで接続されているデータベースを使用している場合、image データ型の処理にはいくつか異なる点があります。『コンポーネント統合サービス・ユーザーズ・ガイド』を参照してください。

「text データ、unitext データ、image データの変更」(235 ページ) を参照してください。

bit データ型

true/false、または yes/no のタイプのデータには、bit カラムを使用します。bit カラムは、0 または 1 のいずれかを保持します。0 または 1 以外の整数値は 1 として解釈されます。記憶サイズは 1 バイトです。テーブル内の複数の bit データ型は、収集されてバイトになります。たとえば 7 つの bit カラムは 1 バイトに納まり、9 つの bit カラムは 2 バイトを取ります。

データ型 bit のカラムは null にはできず、インデックスを持つことはできません。syscolumns システム・テーブルの status カラムは、ビット・カラムのユニークなオフセット位置を示します。

timestamp データ型

Open Client™ DB-Library アプリケーションでブラウズされるテーブルのカラムには、timestamp ユーザ定義データ型が必要です。

timestamp カラムを含むローが挿入または更新されるたびに、timestamp カラムは自動的に更新されます。1 つのテーブルは、timestamp データ型のカラムを 1 つだけ持つことができます。timestamp という名前のカラムは、自動的にシステム・データ型 timestamp を持ちます。定義は次のとおりです。

```
varbinary(8) "NULL"
```

timestamp はユーザ定義データ型なので、他のユーザ定義データ型の定義には使用できません。これは、“timestamp” のようにすべて小文字で入力してください。

sysname データ型および *longsysname* データ型

sysname および longsysname は、システム・テーブルで使用されるユーザ定義データ型です。sysname は次のように定義されます。

```
varchar(30) "NOT NULL"
```

longsysname は、次のように定義されます。

```
varchar(255) "NOT NULL"
```

カラム、パラメータ、または変数を sysname または longsysname 型として宣言できます。また、sysname または longsysname を基本型とするユーザ定義データ型を作成することもできます。

この「ユーザ定義データ型」を使用して、カラムを作成できます。「ユーザ定義データ型の作成」(200 ページ) を参照してください。

Transact-SQL 文における LOB ロケータの使用

ラージ・オブジェクト (LOB) ロケータを使用すると、LOB 自身を参照する代わりに、Transact-SQL 文で LOB を参照することができます。text、unitext、または image の LOB のサイズは数メガバイトになることがあるため、Transact-SQL 文に LOB ロケータを使用することで、クライアントと Adaptive Server 間のネットワーク・トラフィックを低減し、クライアントによる LOB の処理に必要なメモリ量を低減することができます。

Adaptive Server 15.7 では、クライアント・アプリケーションでホスト変数およびパラメータ・マーカとしてロケータを送受信することができます。

LOB ロケータを作成すると、Adaptive Server でメモリ内に LOB 値がキャッシュされ、それを参照する LOB ロケータが生成されます。

LOB ロケータの作成後は、作成されたトランザクションの期間にわたって有効です。Adaptive Server では、トランザクションのコミット時またはロールバック時にロケータが無効になります。

LOB ロケータは 3 種類のデータ型を使用します。

- text_locator − text LOB 用
- unitext_locator − unitext LOB 用
- image_locator − image LOB 用

ロケータ・データ型のローカル変数を宣言できます。次に例を示します。

```
declare @v1 text_locator
```

LOB とロケータはメモリにのみ保存されるため、ロケータのデータ型をユーザのテーブルやビューのカラムのデータ型として、またはデフォルトのデータ型や制約のあるデータ型で使用することはできません。

一般に、Transact-SQL 文で使用される場合は、ロケーションはそれらが参照する LOB に暗黙的に変換されます。つまり、ロケータが Transact-SQL 関数に渡される場合、関数はロケータが参照する LOB で動作します。

ロケータが参照する LOB に対する変更はすべて、明示的に保存しない限り、データベースのソース LOB に反映されるとは限りません。同様に、データベースに格納されている LOB に対する変更はすべて、ロケータが参照する LOB に反映されるとは限りません。

注意 ロケータは、数行のみを返す Transact-SQL 文、またはカーソル文での使用が最適です。これにより、ロケータおよび関連付けられた LOB を処理し、メモリが保持されるように解放することができます。複数の LOB を単一のトランザクションで作成する場合、使用可能なメモリを増加する必要があります。

LOB ロケータの作成

LOB ロケータは明示的または暗黙的に作成することができます。

ロケータを明示的に作成する

create_locator 関数を使用してロケータを明示的に作成します。

- text LOB のためにロケータを作成するには、次を入力します。

  ```
  select create_locator(text_locator, convert(text,
  "some_text_value"))
  ```

- image LOB のロケータを作成するには、次を入力します。

  ```
  select create_locator(image_locator, image_col) from
  table_name
  ```

  たとえば、my_table の image_column カラムに格納されている image LOB のロケータを作成するには、次を入力します。

  ```
  select create_locator(image_locator, image_column) from
  my_table where id=7
  ```

両方の例で、Adaptive Server のメモリに格納されている LOB 値を参照する LOB ロケータを作成して返します。

注意　ロケータを明示的に作成する場合、通常は send_locator 値に関わらずロケータが送信されます。

後続の Transact-SQL 文で使用するために受信したロケータをクライアントのアプリケーションで保存する場合、select 文を使用してロケータを作成するのが最も便利です。isql セッションで、ロケータはローカル変数に割り当てられます。次に例を示します。

```
declare @v text_locator

select @v = create_locator(text_locator, textcol) from
my_table where id = 10
```

注意　空の LOB を参照するロケータを作成することもできます。

ロケータを暗黙的に作成する

1 つのロケータを別のロケータに割り当てる場合、新しいロケータの値は新しい変数に割り当てられます。それぞれのロケータには固有のロケータ値があります。この例では、@v が参照した LOB 値を @w にコピーして作成した新しい LOCATOR を 3 番目の文が割り当てています。次に例を示します。

```
declare @v text_locator, @w text_locator
select @v = create_locator(text_locator, textvol)
from my_table where id = 5
select @w = @v
```

set send_locator on コマンドを使用して、結果セットのすべての LOB 値が関連したロケータ型に変換されてクライアントなどに送信されるよう指定することで、ロケータを暗黙的に作成できます。次に例を示します。

```
set send_locator on
select textcol from my_table where id resulting locators= 5
```

send_locator はオンであるため、textcol の各ローの値に対してロケータが作成され、生成されるロケータはクライアントに送信されます。send_locator がオフ (デフォルト) の場合、実際にテキスト値が送信されます。

ロケータ値を LOB 値に変換する

Transact-SQL 文でロケータを使用したら、ロケータを対応する LOB に変換 (参照外し) できます。

ロケータを明示的に参照外しするには、return_lob 関数を使用します。たとえば、*@w* の LOB 値を返すには、次を入力します。

```
declare @w text_locator

select return_lob(text, @w)
```

ロケータは暗黙的に参照外しすることもできます。たとえば、*@w* の実際の LOB 値を my_table の textcol カラムに挿入するには、次を入力します。

```
insert my_table(textcol) values (@w)
```

注意 return_lob コマンドは set sent_locator on コマンドを無効にし、return_lob は常に LOB を返します。

パラメータ・マーカ

Transact-SQL 文でパラメータ・マーカを使用すると、ロケータを明示的に参照外しできます。次に例を示します。

```
insert my_table (textcol)) values(return_lob(text,?))
```

locator_literal 関数を使用して、次のロケータを識別します。

```
insert my_table (imagecol) values (locator_literal(image_locator,
binary_locator_value))
```

ロケータ・スコープ

通常、ロケータはトランザクションの間有効です。deallocate locator 関数を使用して、デフォルトのスコープを無効にし、トランザクション内でロケータを割り付け解除します。トランザクション内で多くのロケータを作成する必要がある場合、deallocate locator はメモリの節約に特に役立ちます。次に例を示します。

```
begin tran
declare @v text_locator
select @v = textcol from my_table where id=5
deallocate locator @v
...
commit
```

deallocate locator は、トランザクションがコミットする前に *@v* の LOB 値を Adaptive Server のメモリから削除し、ロケータを無効とマーク付けします。

データ型間の変換

Adaptive Server は、あるデータ型から他のデータ型への多数の変換を自動的に処理します。このような自動変換を暗黙的な変換といいます。convert、inttohex、および hextoint 関数を使用して他の変換を明示的に要求することもできます。データ型間に互換性がないために、明示的にも自動的にも実行できない変換もあります。

たとえば Adaptive Server は、char 式を datetime 値として解釈可能で、比較のために自動的に datetime に変換します。ただし、表示のためには、convert 関数を使用して char を int に変換する必要があります。同様に、整数データを like キーワードと使用できるように、Adaptive Server で文字データとして処理する場合は、convert を使用する必要があります。

convert 関数の構文は次のとおりです。

convert (*datatype*, *expression*, [*style*])

次の例では、convert は、数字 2 で開始するすべての売り上げを表示するように、char データ型を使用して total_sales カラムを表示します。

```
select title, total_sales
from titles
where convert(char(20), total_sales) like "2%"
```

さまざまな日付表示フォーマットを得るために datetime 値を char または varchar データ型に変換するために、オプションの style パラメータが使用されます。

convert、inttohex、および hextoint 関数の詳細については、「第 16 章 クエリでの Transact-SQL 関数の使用」を参照してください。

混合モードの算術およびデータ型階層

異なるデータ型の値で算術を実行すると、Adaptive Server はデータ型、および場合によっては結果の長さと精度を、判別する必要があります。

各システム・データ型には、「データ型の階層」があります。この階層は systypes システム・テーブルに保管されています。ユーザ定義データ型は、基になるシステム型の階層を継承します。

次のクエリは、データベースのデータ型を階層でランク付けします。クエリ結果には、ここで示す情報に加えて、データベース内の任意のユーザ定義データ型の情報が含まれます。

```
select name, hierarchy
from systypes
order by hierarchy
```

```
name            hierarchy
---------- ---------
floatn              1
float               2
datetimn            3
datetime            4
real                5
numericn            6
numeric             7
decimaln            8
decimal             9
moneyn             10
money              11
smallmoney         12
smalldatet         13
intn               14
uintn              15
bigint             16
ubigint            17
int                18
uint               19
smallint           20
usmallint          21
tinyint            22
bit                23
univarchar         24
unichar            25
unitext            26
sysname            27
varchar            27
nvarchar           27
longsysnam         27
char               28
nchar              28
timestamp          29
```

```
varbinary         29
binary            30
text              31
image             32
date              33
time              34
daten             35
timen             36
bigdatetim        37
bigtime           38
bigdatetim        39
bigtimen          40
extended t        99
```

注意 *uinteger_type* (usmallint など) は、内部で使用される表現です。符号なし型の正しい構文は、unsigned {int | integer | bigint | smallint } です。

データ型階層は、異なるデータ型の値を使用して計算の結果を判別します。結果の値には、この階層リストの最上位に最も近いデータ型が割り当てられます。

次の例では、sales テーブルの qty に、roysched テーブルの royalty を掛け合わせます。qty は smallint で、階層は 20 です。また、royalty は int で、階層は 18 です。したがって、結果のデータ型は int になります。

```
smallint(qty) * int(royalty) = int
```

次の例では、int (階層は 18) に unsigned int (階層は 19) を掛け合わせます。結果のデータ型は int になります。

```
int(10) * unsigned int(5) = int(50)
```

注意 混合モードの式を使用すると、符号なし整数は、常に符号付きのデータ型に変換されます。符号なし整数値が符号付き整数の範囲内にないときは、変換エラーが発生します。

データ型階層の詳細については、『リファレンス・マニュアル：ビルディング・ブロック』を参照してください。

money データ型での作業

money をリテラルまたは変数と結合していて、money 型の結果が必要な場合は、money リテラルまたは変数を使用します。

```
create table mytable
(moneycol money,)
insert into mytable values ($10.00)
select moneycol * $2.5 from mytable
```

money をカラム値からの float または numeric データ型と結合している場合は、以下のように convert 関数を使用します。

```
select convert (money, moneycol * percentcol)
    from debits, interest
drop table mytable
```

精度と位取りの決定

numeric および decimal 型の場合、精度と位取りの組み合わせは、それぞれ個別の Adaptive Server データ型になります。精度が p1 で位取りが s1 の n1 と、精度が p2 で位取りが s2 の n2 の、2 つの numeric または decimal 値で算術演算を実行する場合、Adaptive Server は 表 6-5 に示すように結果の精度と位取りを決定します。

表 6-5: 算術演算後の精度と位取り

演算	精度	位取り
n1 + n2	max(s1, s2) + max(p1 -s1, p2 - s2) + 1	max(s1, s2)
n1 - n2	max(s1, s2) + max(p1 -s1, p2 - s2) + 1	max(s1, s2)
n1 * n2	s1 + s2 + (p1 - s1) + (p2 - s2) + 1	s1 + s2
n1 / n2	max(s1 + p2 + 1, 6) + p1 - s1 + s2	max(s1 + p2 + 1, 6)

ユーザ定義データ型の作成

SQL への Transact-SQL 拡張機能では、システム・データ型を補う独自のデータ型を設計することが可能です。ユーザ定義データ型はシステム・データ型として定義されます。

注意 1 つのユーザ定義データ型を複数のデータベースで使用するには、model データベース内でデータ型を作成します。こうすると、ユーザ定義データ型の定義は、作成するすべての新しいデータベースで使用できます。

一度定義したデータ型は、データベースの任意のカラムのデータ型として使用できます。たとえば tid は、titles.title_id、titleauthor.title_id、sales.title_id、および roysched.title_id の、いくつかの pubs2 テーブルにおける複数のカラムのデータ型として使用されます。

ユーザ定義データ型の利点は、いくつかのテーブルで使用するためにルールとデフォルトをバインドできることです。「第 14 章 データのデフォルトとルールの定義」を参照してください。

ユーザ・データ型を作成するには、sp_addtype を使用します。これは、作成するデータ型の名前、そのデータ型の基になる Adaptive Server 提供のデータ型、およびオプションの null、not null、または identity の指定をパラメータとしてとります。

timestamp 以外の任意のシステム・データ型を使用して、ユーザ定義データ型を構築できます。ユーザ定義データ型は、基になるシステム・データ型と同じデータ型階層になります。Adaptive Server 提供のデータ型とは異なり、ユーザ定義データ型の名前は大文字と小文字を区別します。

たとえば、データ型 tid を定義するには、次のようにします。

```
sp_addtype tid, "char(6)", "not null"
```

パラメータにブランクや何らかの形式による句読表記が含まれる場合、またはパラメータが null 以外のキーワード (たとえば identity や sp_helpgroup) である場合は、パラメータを一重または二重の引用符で囲んでください。この例では、char(6) にはカッコが含まれ、“not null” にはブランクが含まれているため、それぞれ引用符で囲む必要があります。tid には必要ありません。

長さ、精度、および位取りの指定

場合によっては、次の追加パラメータを指定する必要があります。

- char、nchar、varchar、nvarchar、binary、および varbinary データ型には、カッコ内に長さを指定する。指定しない場合、Adaptive Server はデフォルトの長さである 1 文字とする。
- float 型には、カッコ内に精度を指定する。precision を指定しないと、使用しているプラットフォームのデフォルトの精度が使用されます。
- numeric および decimal データ型には、カッコ内かつカンマ区切りで精度と位取りを指定する。指定しない場合、Adaptive Server はデフォルトの精度 18 と位取り 0 を使用する。

create table 文にユーザ定義データ型を含む場合は、長さ、精度、または位取りを変更できません。

null 型の指定

null 型はユーザ定義データ型がどのように null を処理するかを決定します。“null”、“NULL”、“nonull”、“NONULL”、“not null”、または “NOT NULL” の null 型で、ユーザ定義データ型を作成できます。bit および identity 型は null 値を使用できません。

null 型を省略すると、Adaptive Server は、データベースに定義されている null モードを使用します (デフォルトでは not null)。SQL 規格との互換性のため、sp_dboption を使用して allow nulls by default オプションを true に設定してください。

create table 文にユーザ定義データ型を含む場合は、null 型を上書きできます。

ユーザ定義データ型へのルールとデフォルトの関連付け

ユーザ定義データ型を作成してから、sp_bindrule と sp_bindefault を使用して、データ型にルールとデフォルトを関連付けます。sp_help を使用して、データ型に関連付けられたルール、デフォルト、およびその他の情報をリストするレポートを出力できます。

「第 14 章 データのデフォルトとルールの定義」を参照してください。

IDENTITY プロパティを持つユーザ定義データ型の作成

IDENTITY プロパティを持つユーザ定義データ型を作成するには、sp_addtype を使用します。新しい型は、位取りが 0 の物理型の numeric または任意の整数型に基づいている必要があります。

```
sp_addtype typename, "numeric (precision, 0)", "identity"
```

次の例は、IDENTITY プロパティを持つユーザ定義の型、IdentType を作成します。

```
sp_addtype IdentType, "numeric(4,0)", "identity"
```

IDENTITY 型からカラムを作成するときは、create または alter table 文で identity か not null のいずれかを指定できます。またはいずれも指定しなくてもかまいません。カラムは IDENTITY プロパティを自動的に継承します。

IdentType ユーザ定義型から IDENTITY カラムを作成する、異なる 3 つの方法を次に示します。

```
create table new_table (id_col IdentType)
drop table new_table

create table new_table (id_col IdentType identity)
drop table new_table

create table new_table (id_col IdentType not null)
drop table new_table
```

注意 IDENTITY 型を使用して null の入力可能なカラムを作成しようとすると、create table または alter table 文は失敗します。

ユーザ定義データ型からの IDENTITY カラムの作成

IDENTITY プロパティを持たないユーザ定義データ型から IDENTITY カラムを作成できます。

ユーザ定義データ型は、位取りが 0 の物理データ型の numeric または任意の整数型でなければならず、not null として定義されている必要があります。

ユーザ定義データ型の削除

ユーザ定義データ型を削除するには、sp_droptype を使用します。『リファレンス・マニュアル：プロシージャ』を参照してください。

注意　いずれかのテーブルで使用されているデータ型は、削除できません。

データ型情報の取得

システム・データ型またはユーザ定義データ型のプロパティの情報を表示するには、sp_help を使用します。sp_help からの出力には、データ型の基になる型、null 入力可かどうか、データ型にバインドされているルールとデフォルトの名前、および IDENTITY プロパティを持っているかどうかが含まれています。

次の例は money システム・データ型と tid ユーザ定義データ型についての情報を表示します。

```
sp_help money
Type_name  Storage_type Length Prec  Scale
---------- ------------ ------ ----- -----
money      money             8 NULL  NULL
Nulls   Default_name  Rule_name   Identity
-----   ------------  ---------   --------
     1   NULL          NULL            NULL
(return status = 0)

sp_help tid
Type_name  Storage_type Length Prec  Scale
---------- ------------ ------ ----- -----
tid        varchar           6 NULL  NULL
Nulls   Default_name  Rule_name   Identity
-----   ------------  ---------   --------
    0   NULL          NULL              0

(return status = 0)
```

第 7 章

データの追加、変更、転送、削除

データベース、テーブル、インデックスを作成したら、テーブルにデータを挿入し、必要に応じてデータを操作 (追加、変更、削除) できます。

データの追加、変更、削除に使用するコマンドは、「データ修正文」と呼ばれます。それらのコマンドは以下のとおりです。

- insert − テーブルに新しいローを追加します。
- update − テーブルの既存のローを変更します。
- writetext − システムのトランザクション・ログに大量の変更記録を書き込むことなく、text データ、unitext データ、image データを追加または変更します。
- delete − テーブルから特定のローを削除します。
- truncate table − テーブルからすべてのローを削除します。

『リファレンス・マニュアル：コマンド』を参照してください。

また、テーブルにデータを追加するには、バルク・コピー・ユーティリティ・プログラム bcp を使用してファイルからデータを転送する方法もあります。『ユーティリティ・ガイド』を参照してください。

insert、update、または delete を使用した文ごとに、1 つのテーブルのデータを修正することができます。これらのコマンドに対する Transact-SQL 拡張機能によって、別のテーブルや別のデータベースにあるデータに基づいて修正を行うことができます。

データ修正コマンドもビューに対して機能しますが、いくつかの制限があります。詳細については、「第 12 章 ビュー：データへのアクセスの制限」を参照してください。データベース所有者とデータベース・オブジェクトの所有者は、grant コマンドと revoke コマンドを使用して、データ修正コマンドの実行を許可するユーザを指定できます。

パーミッションや権限は、任意のデータ修正コマンドの組み合わせで個々のユーザ、グループ、または public に付与できます。パーミッションについては、『システム管理ガイド 第 1 巻』の「第 17 章 ユーザ・パーミッションの管理」を参照してください。

参照整合性

insert、update、delete、writetext、および truncate table を使用すると、他のテーブルにある関連データを変更することなくデータを変更できますが、データ間の一貫性が失われることがあります。

たとえば、authors テーブルで Sylvia Panteley に対する au_id エントリを変更した場合は、titleauthor テーブル、およびこの値を含むカラムを持つ、データベース内の他のテーブルすべてにおいて、同じエントリを変更する必要があります。これを実行しないと、Sylvia Panteley の本の書名などの情報を検出することができません。これは、Sylvia Panteley の au_id カラムでジョインを実行できないためです。

データベース内のすべてのテーブルでデータ修正の一貫性を保つことは、「参照整合性」と呼ばれます。参照整合性を管理する方法の 1 つは、テーブルに参照整合性制約を定義することです。もう 1 つの方法は、特定のテーブルやカラムに insert、update、および delete コマンドを実行したときに起動する、トリガと呼ばれる特別なプロシージャを作成することです (truncate table コマンドはトリガまたは参照整合性制約の対象にはなりません)。「第 20 章 トリガ：参照整合性」および「第 8 章 データベースおよびテーブルの作成」を参照してください。

参照整合性テーブルからデータを削除するには、まず参照先テーブルを変更してから、参照元テーブルを変更します。

トランザクション

各データ修正文の影響を受ける各ローの古いステータスと新しいステータスのコピーが、トランザクション・ログに書き込まれます。これは、begin transaction コマンドを実行してトランザクションを開始した後に間違いに気づき、トランザクションをロールバックした場合に、データベースが元の状態にリストアされることを意味します。

注意 リモート・プロシージャ・コール (RPC) によって、リモートの Adaptive Server で行われた変更は、ロールバックできません。

ただし、select/into bulkcopy データベース・オプションが false に設定されている場合は、writetext のデフォルトのモードによるトランザクションのログ記録は行われません。これによって、text、unitext、image フィールドに含まれる、データの非常に長いブロックでトランザクション・ログがいっぱいになることを防ぎます。writetext コマンドで行われる変更をログに記録するには、with log オプションを使用します。

トランザクションの詳細については、「第 23 章 トランザクション：データの一貫性およびリカバリ」で説明します。

サンプル・データベースの使用

この章の例を実行する場合、pubs2 または pubs3 データベースのクリーン・コピーを使用して開始し、終了したらクリーンな状態に戻すことをおすすめします。これらのデータベースのクリーン・コピーの取得については、システム管理者に問い合わせてください。

変更が永続的なものにならないようにするには、入力した文をすべて 1 つのトランザクションに含め、この章を終了したらそのトランザクションをアボートします。たとえば、次のように入力してトランザクションを開始します。

```
begin tran modify_pubs2
```

このトランザクションの名前は modify_pubs2 となります。次のように入力して、トランザクションを任意の時点でキャンセルし、トランザクションを開始する前の状態にデータベースを戻すことができます。

```
rollback tran modify_pubs2
```

データ型の入力の規則

Adaptive Server が提供するデータ型には、ここで説明するように、データの入力や検索についての特別の規則があるものがいくつかあります。データ型については、「第 8 章 データベースおよびテーブルの作成」を参照してください。

char、*nchar*、*unichar*、*univarchar*、*varchar*、*nvarchar*、*unitext*、*text*

character データ、text データ、date データ、time データはすべて、リテラルとして入力するときに、一重引用符か二重引用符で囲む必要があります。set コマンドの quoted_identifier オプションが on に設定されている場合は、一重引用符を使用します。二重引用符を使用すると、Adaptive Server はそのテキストを識別子として扱います。

文字リテラルは、データベースの論理ページ・サイズにかかわらず、任意の長さにすることができます。リテラルが 16 KB (16384 バイト) より長い場合、Adaptive Server はこれを text データとして扱います。text データには、他のデータ型への暗黙および明示的な変換に関する厳密な規則があります。character データ型と text データ型の動作の相違については、『リファレンス・マニュアル：ビルディング・ブロック』の「第 1 章 システム・データ型とユーザ定義データ型」を参照してください。

char、nchar、unichar、univarchar、varchar、または nvarchar カラムに指定された長さより長い文字データを挿入すると、入力はトランケートされます。トランケートが発生した場合は、string_rtruncation オプションを on に設定しておくと、警告メッセージが表示されます。

注意 このトランケーション規則は、カラム、変数、リテラル文字列のいずれに存在するかにかかわらず、すべての文字データに適用されます。

文字エントリにリテラル引用符を指定するには、次の 2 つの方法があります。

- 2 つの引用符を使用する。たとえば一重引用符で文字の入力を始めて、入力の一部に一重引用符を使いたい場合には、一重引用符を 2 つ使い、'I don’t 'understand.' ' のように入力してください。二重引用符の場合は、“He said, “ “It’s not really confusing.” ” ” のようにします。
- 引用符で囲まれている部分を、もう一種の引用符で囲む。つまり、二重引用符を含むエントリを一重引用符で、または一重引用符を含むエントリを二重引用符で囲みます。たとえば、“George said, 'There must be a better way.'” のようにします。

画面の幅より長い文字列を入力するには、次の行に進む前に円記号 (¥) を入力します。

character データ、text データ、datetime データを検索するには、「第 2 章 クエリ：テーブルからのデータの選択」に説明されている like キーワードとワイルドカード文字を使用します。

text データの挿入、および character データの後続ブランクの詳細については、『リファレンス・マニュアル：ビルディング ブロック』の「第 1 章 システム・データ型とユーザ定義データ型」を参照してください。

日付と時刻

Adaptive Server では、datetime、smalldatetime、date、time、bigdatetime、および bigtime のデータ型を使用できます。

日付データと時刻データの表示フォーマットと入力フォーマットは、各種のデータ出力フォーマットを提供し、さまざまな入力フォーマットを認識します。表示フォーマットと入力フォーマットは、別々に制御されます。デフォルトの表示フォーマットでは、“Apr 15 1997 10:23PM” のように出力されます。convert コマンドは、秒とミリ秒を表示し、他の順序で日付部分を表示するためのオプションを提供します。日付値の表示については、「第 16 章 クエリでの Transact-SQL 関数の使用」を参照してください。

Adaptive Server は、日付データのさまざまな入力フォーマットを認識します。大文字と小文字は常に無視され、スペースは日付要素の間のどこにでも入ります。datetime および smalldatetime 値を入力するときは、一重または二重の引用符で囲んでください。quoted_identifier オプションが on に設定されている場合は一重引用符を使用します。二重引用符を使用すると、Adaptive Server は入力を識別子とみなします。

Adaptive Server は、データの日付と時刻の 2 つの部分を別々に認識するので、時刻は日付の前か後に表示できます。いずれの部分も省略できます。その場合、Adaptive Server はデフォルトを使用します。デフォルトの日付と時刻は January 1, 1900, 12:00:00:000AM です。

datetime については、1753 年 1 月 1 日から 9999 年 12 月 31 日までの日付を使用できます。smalldatetime については、1900 年 1 月 1 日から 2079 年 6 月 6 日までの日付を使用できます。bigdatetime については、0001 年 1 月 1 日から 9999 年 12 月 31 日までの日付を使用できます。date については、0001 年 1 月 1 日から 9999 年 12 月 31 日までの日付を使用できます。この範囲より前または後の日付は、char または unichar、あるいは varchar または univarchar の値として入力、格納、操作する必要があります。Adaptive Server は、これらの範囲内の日付として認識できない値をすべて拒否します。

time については、12:00AM から 11:59:59:999 までを使用できます。bigtime については、12:00:00.000000AM から 11:59:59.999999PM までを使用できます。

時刻の入力

時刻のコンポーネントの順序は重要です。まず時間を入力し、次に分、秒、ミリ秒を入力します。そして AM (または am) か PM (pm) を入力します。12AM は夜の 12 時、12PM は昼の 12 時です。時刻として認識されるためには、値にコロンまたは AM か PM の指示子が含まれている必要があります。smalldatetime でカバーされるのは分単位までです。time でカバーされるのはミリ秒単位までです。

ミリ秒はコロンまたはピリオドのどちらに続けてもかまいません。コロンに続ける場合、数字は 1 秒の 1000 分の 1 を意味します。ピリオドに続ける場合、1 桁は 1 秒の 10 分の 1、2 桁は 100 分の 1、3 桁は 1000 分の 1 を意味します。

たとえば、“12:30:20:1” は 12 時 30 分から 20 秒と 1000 分の 1 秒が経過したことを示し、“12:30:20.1” は 12 時 30 分から 20 秒と 10 分の 1 秒が経過したことを示します。

時刻データ用として使用できるフォーマットには、次のものがあります。

```
14:30
14:30[:20:999]
14:30[:20.9]
4am
4 PM
[0]4[:30:20:500]AM
```

bigdatetime および bigtime の表示フォーマットと入力フォーマットにはミリ秒が含まれます。これらのデータ型の時刻は次のように指定する必要があります。

hours[:minutes[:seconds[.microseconds]] [*AM* | *PM*]

hours[:minutes[:seconds[number of milliseconds]] [*AM* | *PM*]

午前 0 時には 12AM を、正午には 12PM を使用してください。bigtime 値には、コロンか AM/PM 指示子を含めてください。AM と PM は大文字でも小文字でも、または両者を組み合わせても入力できます。

秒指定には、小数点の後に小数部分を、またはコロンの後にミリ秒数を含めることができます。たとえば、“12:30:20:1” は午後 12 時 30 分から 20 秒と 1 ミリ秒が過ぎた時刻を示し、“12:30:20.1” は午後 12 時 30 分から 20.1 秒が過ぎた時刻を示します。

ミリ秒が含まれる bigdatetime または bigtime 時刻値を格納するには、小数点を使用したリテラル文字列を指定します。“00:00:00.1” は午前 0 時から 10 分の 1 秒が経過したことを示し、“00:00:00.000001” は午前 0 時から 100 万分の1 秒が経過したことを示します。コロンの後の小数秒を指定する値は、ミリ秒数を示します。たとえば、“00:00:00:5” は 5 ミリ秒を意味します。

日付の入力

set dateformat コマンドは、日付がセパレータ付きの数字の文字列として入力される場合の、日付要素の順序 (月、日、年) を指定します。set language を使用すると、指定した言語のデフォルトの日付フォーマットによって、日付のフォーマットにも影響する場合があります。デフォルト言語は us_english で、デフォルトの日付フォーマットは *mdy* です。『リファレンス・マニュアル：コマンド』を参照してください。

注意 dateformat は、“4/15/90” や “20.05.88” といった、セパレータ付きの数字として入力された日付だけに影響します。“April 15, 1990” のように月がアルファベットで入力されているものや、“19890415” のようにセパレータのないものには、影響しません。

日付フォーマット

Adaptive Server は、次に示す 3 つの基本的な日付フォーマットを認識します。各フォーマットは引用符で囲む必要があり、「時刻の入力」(209 ページ) で説明する時刻指定の前または後に表示できます。

- 月をアルファベットで入力する。
  - 日付をアルファベットで指定する場合の有効なフォーマットは次のとおりです。

```
Apr[il] [15][,] 1997
Apr[il] 15[,] [19]97
Apr[il] 1997 [15]
[15] Apr[il][,] 1997
15 Apr[il][,] [19]97
15 [19]97 apr[il]
[15] 1997 apr[il]
1997 APR[IL] [15]
1997 [15] APR[IL]
```

  - 月は、現在の言語の仕様に従って、3 文字の省略形か、フルスペルで表示できます。
  - カンマはオプションです。
  - 大文字と小文字は無視されます。
  - 西暦年の下 2 桁だけを指定した場合、50 未満の値は “20yy”、50 以上の値は “19yy” と解釈されます。
  - 日を省略する場合、またはデフォルト以外の西暦年の上 2 桁を指定する場合は、西暦年の上 2 桁を入力します。
  - 日が指定されていないと、Adaptive Server はデフォルトでその月の最初の日を使用します。
  - 月をアルファベットで指定すると、dateformat の設定は無視されます (『リファレンス・マニュアル：コマンド』を参照してください)。
- 月を、スラッシュ (/)、ハイフン (-)、またはピリオド (.) のセパレータを使用した文字列で、数値で入力する。
  - 月、日、および年を指定する必要があります。
  - 文字列は、次の形式である必要があります。

    <num> <sep> <num> <sep> <num> [<time spec>]

    または

    [<time spec>] <num> <sep> <num> <sep> <num>

- 日付要素の値の解釈は、dateformat 設定によって異なります。順序が設定と一致しない場合、値は、範囲外にあるために日付として解釈されないか、誤って解釈されます。たとえば、“12/10/08” は、dateformat の設定によって、異なる 6 つの日付の 1 つとして解釈される可能性があります。『リファレンス・マニュアル：コマンド』を参照してください。
- *mdy* dateformat で “April 15, 1997” を入力するには、次のフォーマットを使用します。

```
[0]4/15/[19]97
[0]4-15-[19]97
[0]4.15.[19]97
```

- これ以外の入力順を、/ をセパレータとして次に示します。セパレータにはハイフンやピリオドを使用することもできます。

```
15/[0]4/[19]97 (dmy)
1997/[0]4/15 (ymd)
1997/15/[0]4 (ydm)
[0]4/[19]97/15 (myd)
15/[19]97/[0]4 (dym)
```

- セパレータを使用しない 4 桁、6 桁、または 8 桁の文字列で指定する。または空文字列、つまり日付値を指定しないで時刻値だけを指定する。
  - dateformat はこの入力フォーマットを常に無視します。
  - 4 桁を指定した場合、文字列は西暦年として解釈され、月は 1 月に、日はその月の最初の日に設定されます。西暦年の上 2 桁は省略できません。
  - 6 桁または 8 桁の文字列は、常に *ymd* として解釈されます。月日は 2 桁でなければなりません。[19]960415 というフォーマットは認識されます。
  - 空文字列 (“”) が指定されるか、日付が指定されない場合は、基本の日付である 1900 年 1 月 1 日として解釈されます。たとえば、日付を指定しない “4:33” という時刻値は、「1900 年 1 月 1 日午前 4 時 33 分」と解釈されます。

set datefirst コマンドは、weekday または dw を datename とともに使用した場合は曜日 (日曜日、月曜日など) を、datepart とともに使用した場合は対応する数字を指定します。set language を使用して言語を変更すると、その言語のデフォルトの最初の曜日の値によって、日付のフォーマットに影響します。us_english のデフォルト言語の場合、デフォルトの datefirst 設定は Sunday=1、Monday=2 というようになっています。それ以外では Monday=1、Tuesday=2 などとなります。set datefirst を使用すると、セッションごとにデフォルトの動作を変更できます。『リファレンス・マニュアル：コマンド』を参照してください。

日付と時刻の検索

like キーワードとワイルドカード文字は、char、unichar、nchar、varchar、univarchar、nvarchar、text、unitext と同様に、datetime、smalldatetime、bigdatetime、bigtime、date、time データにも使用できます。like を date 値および time 値とともに使用すると、Adaptive Server は、日付をまず標準の date/time フォーマットに変換してから、varchar または univarchar に変換します。datetime と smalldatetime の標準の表示フォーマットには秒やミリ秒は含まれていないため、like と一致パターンを使用して秒やミリ秒を検索することはできません。秒およびミリ秒を検索するには、データ型変換関数 convert を使用します。

datetime、bigtime、bigdatetime、smalldatetime などのエントリにはさまざまな日付要素が含まれている可能性があるので、これらのデータ型の値を検索するときは like を使用します。たとえば、arrival_time というカラムに値 “9:20” を挿入すると、Adaptive Server ではこのエントリが “Jan 1, 1900 9:20AM” に変換されるため、次の句ではこの値を検出できません。

```
where arrival_time = "9:20"
```

しかし、次の句では検出できます。

```
where arrival_time like "%9:20%"
```

これは、date データ型と time データ型を使用しているときにも当てはまります。

like を使用していて、日が 10 を下回る場合は、月と日の間にスペースを 2 つ挿入して、datetime 値の varchar 変換に対応するようにしてください。同様に、時が 10 を下回る場合は、変換によって年と時の間にスペースが 2 つ置かれます。“May” と “2” の間にスペースを 1 つ持つ like May 2% 句は、May 20 から May 29 までの日付をすべて検出しますが、May 2 は検出しません。datetime 値は、like 比較の場合のみ varchar に変換されるので、like 比較以外の日付比較では余分なスペースを入力する必要はありません。

binary、*varbinary*、および *image*

binary、varbinary、または image のデータをリテラルとして入力するときは、データの前に “0x” を付けます。たとえば、“FF” と入力するには “0xFF” と入力します。ただし、“0x” で始まるデータは引用符で囲まないでください。

バイナリ・リテラルは、データベースの論理ページ・サイズにかかわらず、任意の長さにすることができます。リテラルの長さが 16 KB (16384 バイト) 未満の場合、Adaptive Server はリテラルを varbinary データとして扱います。リテラルの長さが 16 KB を超える場合、Adaptive Server はそのリテラルを image データとして扱います。

binary データを、指定された長さがデータ長よりも短いカラムに挿入すると、エントリは警告なしでトランケートされます。

binary または varbinary カラムの長さ 10 は 10 バイトを示し、それぞれ 2 桁の 16 進数を格納します。

binary または varbinary カラムにデフォルトを作成するときは、その前に “0x” を付けてください。

binary データ型と image データ型の動作の相違、および 16 進の値の後続のゼロについては、『リファレンス・マニュアル：ビルディング・ブロック』の「第 1 章 システム・データ型とユーザ定義データ型」を参照してください。

money および *smallmoney*

E 表記を使用して入力された通貨値は、float として解釈されます。このため、値が money または smallmoney 値として格納されると、入力が拒否されたり、精度が若干失われたりすることがあります。

money および smallmoney 値を入力するときには、ドル記号 ($)、円記号 (¥)、通貨ポンド記号 (£) などの通貨記号を数値の前に付けても、付けなくてもかまいません。負の値を入力する場合には、通貨記号の後にマイナス符号を入力してください。入力にはカンマは含めないでください。

money データや smallmoney データのデフォルトの出力フォーマットでは 3 桁ごとにカンマを使用していますが、money 値や smallmoney 値を、カンマを使用して入力することはできません。money または smallmoney 値は、表示されるときに最も近い補助貨幣の値に丸められます。money では、モジュロを除くすべての算術演算が使用可能です。

float、*real*、および *double precision*

オプションの指数の前には、概数値型 (float、real、double precision) を「仮数」として入力します。仮数部には正か負の記号および小数点を含めることができます。文字 “e” または “E” の後から始まる指数には、記号を含めることができますが、小数点を含めることはできません。

概数値データを見積もるために、Adaptive Server は、指定された指数による 10 の累乗を仮数部に乗じます。表 7-1 は、float、real、double precision のデータの例を示します。

表 7-1: 数値データの見積もり

入力データ	仮数部	指数	値
10E2	10	2	$10 * 10^2$
15.3e1	15.3	1	$15.3 * 10^1$
-2.e5	-2	5	$-2 * 10^5$
2.2e-1	2.2	-1	$2.2 * 10^{-1}$
+56E+2	56	2	$56 * 10^2$

カラムのバイナリ精度は、仮数部に使用できるバイナリ桁数の最大数を決定します。float カラムの場合は、48 桁までの精度を指定できます。real および double precision カラムの場合は、精度はマシンに依存します。値がカラムのバイナリ精度を超えると、Adaptive Server は、そのエントリをエラーとして通知します。

decimal および *numeric*

真数値型 (dec、decimal、numeric) は、オプションの正か負の記号で開始され、小数点を含めることができます。真数値データの値は、カラムの 10 進の *precision* および *scale* に依存します。これは次の構文を使用して指定します。

datatype [(*precision* [, *scale*])]

Adaptive Server は、精度と位取りの各組み合わせを別個のデータ型として扱います。たとえば numeric(10,0) と numeric(5,0) は、2 つの別々のデータ型です。精度と位取りは、decimal または numeric カラムに格納できる値の範囲を決定します。

- 精度は、カラムに格納できる 10 進の桁の最大数を指定します。小数点の左右のすべての桁がこれに含まれます。1 から 38 桁までの範囲の精度を指定できます。または 18 桁のデフォルトの精度を使用できます。
- 位取りは、小数点の右側に格納できる桁数の最大数を指定します。位取りは精度以下でなければならない。0 から 38 桁までの位取りを指定するか、またはデフォルトの位取りである 0 桁を使用します。

値がカラムの精度または位取りを超えると、Adaptive Server は、そのエントリをエラーとして通知します。有効な dec および numeric データの例をいくつか示します。

表 7-2: 数値データに有効な精度と位取り

入力データ	データ型	精度	位取り	値
12.345	numeric(5,3)	5	3	12.345
-1234.567	dec(8,4)	8	4	-1234.567

次のエントリは、カラムの精度または位取りを超えるため、エラーになります。

表 7-3: 数値データに無効な精度と位取り

入力データ	データ型	精度	位取り
1234.567	numeric(3,3)	3	3
1234.567	decimal(6)	6	1

符号付き整数型と符号なし整数型

前述の項で説明したように、bigint、int、smallint、tinyint、unsigned bigint、unsigned int、unsigned smallint カラムに、E 表記を使用して数値を挿入できます。

timestamp

timestamp カラムにはデータを挿入できません。カラムに “null” と入力して明示的な null を挿入するか、timestamp カラムを省略したカラム・リストを提供して暗黙的な null を使用する必要があります。Adaptive Server は、挿入または更新があるたびに、timestamp 値を更新します。「特定のカラムへのデータの挿入」(217 ページ) を参照してください。

新しいデータの追加

insert コマンドを使用すると、次の 2 通りの方法でデータベースにローを追加できます。

- values キーワードを使用して、新しいローにあるカラムのいくつか、またはすべてに新しい値を指定します。values キーワードを使用した insert コマンドの構文の簡易バージョンを次に示します。

  ```
  insert table_name
      values (constant1, constant2, ...)
  ```

- insert 文の中で select 文を使用すると、1 つ以上のテーブル (挿入を行うテーブルを含め、最大 50 テーブル) から値を取得できます。select 文を使用した insert コマンドの構文の簡易バージョンを次に示します。

  ```
  insert table_name
      select column_list
      from table_list
      where search_conditions
  ```

注意 compute を含む文は通常のローを生成しないため、insert 文内の select 文で compute 句を使用することはできません。

insert を使用して text 値、unitext 値、image 値を追加すると、すべてのデータがトランザクション・ログに書き込まれます。writetext コマンドを使用すると、text、unitext、image 値を含む可能性のある長いデータをログに記録することなく、これらの値を追加できます。詳細については、「特定のカラムへのデータの挿入」(217 ページ) と「text データ、unitext データ、image データの変更」(235 ページ) を参照してください。

values による新しいローの追加

次の insert 文は、publishers テーブルに新しいローを追加し、そのローのすべてのカラムに値を追加します。

```
insert into publishers
values ("1622", "Jardin, Inc.", "Camden", "NJ")
```

元の create table 文のカラム名と同じ順、つまり、最初に ID 番号、次に名前、都市、そして州という順で、データ値が入力されています。values データはカッコで囲まれており、文字データはすべて一重または二重の引用符で囲まれています。

追加するローごとに個別の insert 文を使用します。

特定のカラムへのデータの挿入

ローのいくつかのカラムとそのデータだけを指定することによって、そのカラムにデータを追加できます。カラム・リストに含まれていない他のカラムは、すべて null 値を許可するように定義されている必要があります。省略されるカラムはデフォルトを使用できます。デフォルトがバインドされているカラムを省略すると、デフォルトが使用されます。

この形式の insert コマンドは、text 値、unitext 値、image 値以外のすべての値をローに挿入し、これらの値がトランザクション・ログに格納されないように、writetext を使用して長いデータ値を挿入するのに便利です。また、この形式のコマンドを使用して、timestamp データを省略することもできます。

2 つのカラムだけにデータを追加するには、次のようなコマンドが必要です。pub_id と pub_name を例とします。

```
insert into publishers (pub_id, pub_name)
values ("1756", "The Health Center")
```

カラム名をリストする順序は、値をリストする順序と一致している必要があります。次の例は、前述のものと同じ結果を生成します。

```
insert publishers (pub_name, pub_id)
values("The Health Center", "1756")
```

いずれの insert 文も、ID 番号カラムに “1756” を挿入し、出版社名カラムに “The Health Center” を挿入します。publishers の pub_id カラムにはユニーク・インデックスがあるため、これらの insert 文を両方とも実行することはできません。pub_id 値 “1756” を 2 度目に挿入しようとすると、エラー・メッセージが生成されます。

次の select 文は、publishers に追加されたローを表示します。

```
select *
from publishers
where pub_name = "The Health Center"

pub_id  pub_name            city    state
```

```
------- -----------------    ------  -------
1756    The Health Center    NULL    NULL
```

Adaptive Server は、city カラムおよび state カラムに null 値を入力します。これらのカラムには insert 文で値が指定されておらず、publisher テーブルではこれらのカラムに null 値が許可されているためです。

カラム・データの制限：ルール

ルールを作成して、カラムやユーザ定義データ型にバインドできます。ルールは追加できるデータまたは追加できないデータの種類を制御します。

たとえば、pub_idrule という、受け入れ可能な出版社の ID 番号を指定するルールが、publishers テーブルの pub_id カラムにバインドされているとします。受け入れ可能な ID は、“1389”、“0736”、“0877”、“1622”、“1756” のいずれか、または “99” で始まる任意の 4 桁の数値です。これ以外の数値を入力すると、エラー・メッセージが表示されます。

この種のエラー・メッセージが表示された場合は、sp_helptext を使用して、ルールの定義を参照できます。

```
sp_helptext pub_idrule
---------
        1

(1 row affected)

text
--------------------------------------------------
create rule pub_idrule
as @pub_id in ("1389", "0736", "0877", "1622", "1756")
or @pub_id like "99[0-9][0-9]"

(1 row affected)
```

特定のルールに関する一般的な説明については、sp_help を使用してください。また、カラムにルールがあるかどうかを知るには、パラメータとしてテーブル名を使用して sp_help を使用してください。「第 14 章 データのデフォルトとルールの定義」を参照してください。

null 文字列の使用

create table 文内で null が指定され、ユーザが明示的に null を (引用符を付けずに) 挿入したカラム、またはデータが挿入されていないカラムだけが、null 値を含みます。文字カラムには、文字列 “null” (引用符付き) をデータとして入力しないでください。代わりに “N/A” や “none” などの値を使用してください。

カラムに null を明示的に挿入するには、次の構文を使用します。

```
values({expression | null}
    [, {expression | null}]...)
```

次の例に、等価な 2 つの insert 文を示します。最初の文では、ユーザはカラム t1 に明示的に null を挿入しています。2 番目の文では、ユーザが明示的にカラム値を指定しなかったため、Adaptive Server が t1 に null を渡しています。

```
create table test
(t1 char(10) null, t2 char(10) not null)
insert test
values (null, "stuff")
insert test (t2)
values ("stuff")
```

null は空文字列ではない

空文字列 (“ ” や ‘ ’) は、常にシングル・スペースとして変数やカラム・データに格納されます。次の連結文は、“abc def” と等価ですが、“abcdef” とは等価ではありません。

```
"abc" + "" + "def"
```

空文字列は null として評価されることはありません。

null を許可しないカラムへの null の挿入

select を使用して、null 値を持つフィールドがあるテーブルから、null 値を許可しないテーブルへデータを挿入するには、元のテーブルの null エントリに値を代入する必要があります。たとえば、次の例は、null 値を許可しない advances テーブルにデータを挿入するために、null フィールドに “0” を代入します。

```
insert advances
select pub_id, isnull(advance, 0) from titles
```

isnull 関数を指定しないと、このコマンドは非 null 値を持つすべてのローを advances に挿入し、titles の advance カラムに null が含まれているすべてのローに対してエラー・メッセージを生成します。

データにこのような代入を実行できない場合は、not null が指定されているカラムに、null 値を含むデータを挿入することはできません。

すべてのカラムに値を持たないローの追加

ローのカラムのいくつかだけに値を指定すると、値のないカラムは次のいずれかになります。

- カラムまたはカラムのユーザ定義データ型にデフォルト値がある場合は、そのデフォルト値が入力されます。詳細については、「第 14 章 データのデフォルトとルールの定義」、または『リファレンス・マニュアル：コマンド』の「insert」を参照してください。

- テーブルの作成時にカラムに null が指定され、カラムまたはデータ型にデフォルト値が存在しない場合は、null が挿入されます。詳細については、『リファレンス・マニュアル：コマンド』の「insert」を参照してください。
- カラムに IDENTITY プロパティがある場合は、ユニークな連続する値が入力されます。
- テーブル作成時にカラムに null が指定されておらず、デフォルトが存在しない場合、Adaptive Server は、そのローを拒否してエラー・メッセージを表示します。

表 7-4 に、このような状況の結果を示します。

表 7-4: 値を持たないカラム

カラムまたはデータ型にデフォルトが存在する	カラムが not null と定義されている	カラムが null を許可するように定義されている	カラムが IDENTITY である
はい	デフォルト	デフォルト	次の連続する値
いいえ	エラー・メッセージ	null	次の連続する値

sp_help を使用すると、指定するテーブルまたはデフォルトに関するレポートや、システム・テーブル sysobjects にリストされているその他のオブジェクトに関するレポートを表示できます。デフォルトの定義を参照するには、sp_helptext を使用します。

カラム値の null への変更

カラム値を null に設定するには、update 文を次のように使用します。

```
set column_name = {expression | null}
[, column_name = {expression | null}]...
```

次の例は、title_id が TC3218 であるローをすべて検出し、advance を null で置き換えます。

```
update titles
set advance = null
where title_id = "TC3218"
```

Adaptive Server によって生成される IDENTITY カラムの値

IDENTITY カラムを持つテーブルにローを挿入すると、Adaptive Server は自動的にカラム値を生成します。IDENTITY カラムの名前をカラム・リストに含めたり、その値を値リストに含めたりしないでください。

次の insert 文は、sales_daily テーブルに新しいローを追加します。カラム・リストには、IDENTITY カラム、row_id は含まれません。

```
insert sales_daily (stor_id)
values ("7896")
```

注意 次の例では、カラム名 stor_id も省略できます。サーバは、ユーザがカラム名を入力しなくても、IDENTITY カラムを識別し、次の identity 値を挿入できます。たとえば、このテーブルには 3 つのカラムがありますが、insert 文は 2 つのカラムの値を指定し、カラム名は指定しません。

```
create table idtext (a int, b numeric identity, c char(1))
-------------------
(1 row affected)

insert idtext values(98,"z")
-------------------
(1 row affected

insert idtest values (99, "v"))
--------------------

(1 row affected)
select * from idtest
---------------------
98     1         z
99     2          v

(2 rows affected)
```

次の文は、sales_daily に追加されたローを表示します。Adaptive Server は、row_id に、次の連続する値 2 を自動的に生成します。

```
select * from sales_daily
where stor_id = "7896"

sale_id      stor_id
-------      -------
      1      7896

(1 row affected)
```

IDENTITY カラムへの明示的なデータ挿入

場合によっては、IDENTITY カラムに特定の値を挿入する必要がある場合もあります。たとえば、テーブルに挿入された最初のローが、1 ではなく 101 という IDENTITY 値を持つようにする場合や、誤って削除されたローを挿入し直す必要がある場合などです。

テーブル所有者は、IDENTITY カラムに明示的に値を挿入できます。データベース所有者およびシステム管理者は、テーブル所有者によって明示的にパーミッションを付与されている場合、またはテーブル所有者として操作を行っている場合に、IDENTITY カラムに明示的に値を挿入できます。

データを挿入する前に、テーブルに対して identity_insert オプションを on に設定します。セッション中の 1 つのデータベースでは、一度に 1 つのテーブルにだけ identity_insert を on に設定できます。

次の例は、IDENTITY カラムに “seed” 値 101 を指定します。

```
set identity_insert sales_daily on
insert sales_daily (syb_identity, stor_id)
values (101, "1349")
```

insert 文では、IDENTITY カラムを含めて各カラムをリストして、それぞれに値を指定します。identity_insert オプションが on に設定される場合、テーブルに対する各 insert 文は明示的なカラム・リストを指定する必要があります。IDENTITY カラムでは null 値の使用が許可されていないので、値リストは IDENTITY カラム値を指定する必要があります。

identity_insert を off に設定した後は、IDENTITY カラムを指定せずに IDENTITY カラムの値を自動的に挿入できます。以降の挿入では、identity_insert を on に設定した後に明示的に指定された値に基づいた IDENTITY 値を使用します。たとえば IDENTITY カラムに 101 を指定すると、以降の挿入は 102、103 というようになります。

注意 Adaptive Server は、挿入される値のユニーク性を強制しません。宣言されたカラムの精度で許可されている範囲内の、任意の正の整数を指定できます。ユニークなカラム値だけを受け入れるようにするには、IDENTITY カラムにユニーク・インデックスを作成してから、ローを挿入します。

@@*identity* を使用した IDENTITY カラム値の取得

IDENTITY カラムに挿入された最新の値を取得するには、*@@identity* グローバル変数を使用します。*@@identity* の値は、insert または select into がテーブルにローを挿入しようとするたびに変更されます。insert または select into 文が失敗した場合、またはそれを含むトランザクションがロールバックされた場合、*@@identity* は以前の値には戻りません。文が IDENTITY カラムを持たないテーブルに影響する場合、*@@identity* は 0 に設定されます。

文が複数のローを挿入する場合、*@@identity* は、IDENTITY カラムに最後に挿入された値を反映します。

ストアド・プロシージャやトリガの中にある *@@identity* の値は、ストアド・プロシージャやトリガの外側にある値には影響しません。次に例を示します。

```
select @@identity

---------------------------------------
                                   101

create procedure reset_id as
    set identity_insert sales_daily on
    insert into sales_daily (syb_identity, stor_id)
```

```
         values (102, "1349")
    select @@identity
select @@identity
execute reset_id

---------------------------------------
                                    102

select @@identity

---------------------------------------
                                    101
```

IDENTITY カラム値のブロックの予約

identity grab size 設定パラメータによって、各 Adaptive Server は、IDENTITY カラムを持つテーブルへの挿入用に IDENTITY カラム値のブロックを予約する処理を行うことができます。この設定パラメータは、暗黙的な identity 値を挿入するときに Adaptive Server エンジンが内部同期構造を保持しなければならない時間を短縮します。たとえば、次のコマンドは、予約値の数を 20 に設定します。

```
sp_configure "identity grab size", 20
```

ユーザが IDENTITY カラムを含むテーブルに挿入を実行すると、Adaptive Server は、20 個の IDENTITY カラム値のブロックをそのユーザに予約します。このため、現在のセッションでは、ユーザがテーブルに挿入する次の 20 個のローは、連続する IDENTITY カラム値を持つことになります。最初のユーザが挿入を実行している間に、次のユーザが同じテーブルにローを挿入すると、Adaptive Server は、次の 20 個の IDENTITY カラム値のブロックを 2 番目のユーザ用に予約します。

たとえば、IDENTITY カラムを持つ次のテーブルが作成され、identity grab size が 10 に設定されているとします。

```
create table my_titles
(title_id   numeric(5,0)    identity,
title       varchar(30)     not null)
```

user 1 は、これらのローを my_titles テーブルに挿入します。

```
insert my_titles (title)
values ("The Trauma of the Inner Child")insert my_titles
(title)

values ("A Farewell to Angst")
insert my_titles (title)
values ("Life Without Anger")
```

Adaptive Server は、たとえば、title_id 番号 1 ～ 10 のように、連続する 10 個の IDENTITY 値のブロックを user 1 に許可します。

user 1 が my_titles にローを挿入している間に、user 2 が my_titles にローの挿入を開始します。Adaptive Server は、user 2 に対し、予約済み IDENTITY 値の、次に使用可能なブロック、つまり、値 11 ～ 20 を許可します。

user 1 がタイトルを 3 つだけ入力して Adaptive Server からログオフすると、残りの 7 つの予約済み IDENTITY 値は失われます。結果として、テーブルの IDENTITY 値にギャップが発生します。IDENTITY カラムのギャップが大きくならないようにするために、identity grab size の値にあまり高い値を指定しないでください。

IDENTITY カラムの最大値を超えた場合

IDENTITY カラムに挿入できる最大値は、$10^{precision} - 1$ です。IDENTITY カラムに精度を指定しない場合、Adaptive Server は、numeric カラムにデフォルトの精度 (18 桁) を使用します。

IDENTITY カラムがその最大値に達すると、挿入文は、現在のトランザクションをアボートするエラーを返します。このような状況が発生した場合は、次のいずれかの方法で問題を解決します。

IDENTITY カラムの最大値を変更する

alter table コマンドで変更オペレーションを使用すると、IDENTITY カラムの最大値を変更できます。

```
alter table my_titles
modify title_id, numeric (10,0)
```

このオペレーションは、テーブルへのデータ・コピーを実行し、テーブル・インデックスをすべて再構築します。

より大きい精度を持つ新しいテーブルを作成する

テーブルに、参照整合性に使用する IDENTITY カラムが含まれる場合は、IDENTITY カラム値の現在の番号を保持してください。

1 create table を使用して、以前のテーブルの IDENTITY カラムの精度の値を大きくしたテーブルを新しく作成します。

2 insert into を使用して、以前のテーブルから新しいテーブルにデータをコピーします。

bcp を使用してテーブルの IDENTITY カラムの番号を付け直す

テーブルに、参照整合性に使用する IDENTITY カラムが含まれておらず、番号付けシーケンスにギャップが発生している場合は、IDENTITY カラムの再番号付けを行ってギャップを解消できます。これによって、より多くの領域を挿入に使用できます。

IDENTITY カラム値を連続した番号に付け直す (それによってギャップを解消する) には、次の手順に従います。

1　オペレーティング・システムのコマンド・ラインから bcp を使用して、データをコピー・アウトします。

```
bcp pubs2..mytitles out my_titles_file -N -c
```

-N は IDENTITY カラム値をテーブルからホスト・ファイルにコピーしないように、-c は、文字モードを使用するように、bcp にそれぞれ指示します。

2　Adaptive Server で、以前のテーブルと同一の新しいテーブルを作成します。

3　オペレーティング・システムのコマンド・ラインから bcp を使用して、新しいテーブルにデータをコピー・インします。

```
bcp pubs2..mynewtitles in my_titles_file -N -c
```

-N は、Adaptive Server がホスト・ファイルからデータをロードするときに IDENTITY カラム値への代入を行うように、-c は、文字モードを使用するように bcp にそれぞれ指示します。

4　Adaptive Server で、以前のテーブルを削除し、sp_rename を使用して、新しいテーブルの名前を以前のテーブルの名前に変更します。

IDENTITY カラムがジョインのプライマリ・キーである場合は、他のテーブルの外部キーを更新する必要があります。

デフォルトでは、ユーザが IDENTITY カラムのあるテーブルにデータをバルク・コピーするときに、bcp は、各ローに、テンポラリの IDENTITY カラム値 0 を割り当てます。bcp がテーブルに各ローを挿入するときには、サーバは、次に使用可能な値で始まるユニークで連続した IDENTITY カラム値を割り当てます。各ローに明示的な IDENTITY カラム値を入力するには、-E フラグを指定します。『ユーティリティ・ガイド』を参照してください。

select による新しいローの追加

1 つ以上の他のテーブルからテーブルに値を取得するには、insert 文で select 句を使用します。select 句は、ローのいくつかのカラム、またはすべてのカラムに値を挿入できます。

既存のテーブルから値を取得する場合は、一部のカラムだけに値を挿入すると便利です。その後、update を使用して、他のカラムに値を追加できます。

テーブル内のすべてのカラムではなくいくつかのカラムだけに値を挿入する場合は、値を挿入しないカラムにデフォルトが存在するか、または null が指定されていることを事前に確認します。そうしないと、Adaptive Server はエラー・メッセージを返します。

あるテーブルから別のテーブルへローを挿入する場合は、その 2 つのテーブルが互換性のある構造を持っている必要があります。つまり、対応するカラムが同じデータ型、または Adaptive Server が自動変換するデータ型でなければなりません。

注意 挿入されるデータのいずれかが null の場合、null 値の入力を許可するテーブルから、null 値の入力を許可しないテーブルへは、データを挿入できません。

カラムが create table 文内と同じ順序である場合は、いずれのテーブルでもカラム名を指定する必要はありません。newauthors という名前のテーブルがあり、authors と同じフォーマットで作家情報のローがいくつか含まれているとします。newauthors のすべてのローを authors に追加するには、次を実行します。

```
insert authors
select *
from newauthors
```

あるテーブルに、別のテーブルのデータに基づいてローを挿入するには、2 つのテーブルのカラムがそれぞれの create table 文で同じ順にリストされている必要はありません。insert または select 文のいずれかを使用して、カラムが一致するよう順序付けることができます。

たとえば、authors テーブルに対する create table 文に、カラム au_id、au_fname、au_lname、address がこの順序で含まれていて、au_id、address、au_lname、au_fname を含む newauthors があるとします。カラム・シーケンスは insert 文内のものと一致させる必要があります。これは、次の手順で行います。

```
insert authors (au_id, address, au_lname, au_fname)
select * from newauthors
```

または

```
insert authors
select au_id, au_fname, au_lname, address
    from newauthors
```

2 つのテーブルのカラム・シーケンスが一致しない場合、Adaptive Server は、insert オペレーションを完了できないか、誤ったカラムにデータを挿入して、誤った形で完了します。たとえば、au_lname カラムに住所のデータが挿入されたりします。

計算カラムの使用

insert 文の中で、select 文に計算カラムを使用できます。たとえば、tmp という名前のテーブルに、titles テーブルの新しいローがいくつか含まれているとします。titles テーブルには古いデータが含まれており、price の数字を倍にする必要があります。価格を上げ、titles に tmp ローを挿入する文は、次のようになります。

```
insert titles
select title_id, title, type, pub_id, price*2,
    advance, total_sales, notes, pubdate, contract
from tmp
```

カラムで計算を実行する場合、select * 構文は使用できません。各カラムは select リストで個別に指定する必要があります。

複数のカラムへのデータの挿入

select 文を使用すると、ローのすべてのカラムではなく、一部のカラムにデータを追加できます。insert 句で、データを追加したいカラムを指定するだけです。

たとえば、authors テーブルにある作家のなかには、タイトルがないために titleauthor テーブルにエントリのないものがあります。このような作家の au_id 番号を authors テーブルから取り出し、プレースホルダとして titleauthor テーブルに挿入するために、次の文を実行してみます。

```
insert titleauthor (au_id)
select au_id
    from authors
    where au_id not in
    (select au_id from titleauthor)
```

title_id カラムに値が必要なため、この文は正しくありません。null 値の入力は許可されておらず、デフォルトも指定されていません。次のように定数を使用して、titles_id にダミー値 “xx1111” を入力できます。

```
insert titleauthor (au_id, title_id)
select au_id, "xx1111"
    from authors
    where au_id not in
    (select au_id from titleauthor)
```

これで、titleauthor テーブルには、au_id カラムにエントリ、title_id カラムにダミー・エントリ、およびその他 2 つのカラムに null 値を持つ、4 つの新しいローが含まれました。

同じテーブルからのデータの挿入

テーブルに、同じテーブルの別のデータに基づいて、データを挿入できます。つまり、これはあるローのすべてまたは一部をコピーすることを意味します。

たとえば、次のように、publishers テーブルに、同じテーブルの既存のローの値に基づいた新しいローを挿入できます。pub_id カラムのルールに従っていることを確認してください。

```
insert publishers
select "9999", "test", city, state
    from publishers
    where pub_name = "New Age Books"

 (1 row affected)

select * from publishers

 pub_id  pub_name              city        state
------- --------------------- -------     ------
0736    New Age Books         Boston      MA
0877    Binnet & Hardley      Washington  DC
1389    Algodata Infosystems  Berkeley    CA
9999    test                  Boston      MA

(4 rows affected)
```

この例では、2 つの定数 (“9999” と “test” と、クエリを満たすローの city カラムおよび state カラムの値を挿入します。

マテリアライズされていない非 null カラムの作成

マテリアライズされていないカラムは仮想的に存在しますが、ロー内に物理的に格納されるわけではありません。マテリアライズされていないカラムは、選択、更新、SQL クエリでの参照、インデックス・キーとしての使用で、他のカラムと同じように使用します。

Adaptive Server では、マテリアライズされていないカラムが、null カラムと同様に処理されます。カラムがロー内に物理的に存在しない場合は、Adaptive Server によりデフォルトのローが提供されます。null 入力可能なカラムのデフォルトは `null` ですが、マテリアライズされていないカラムのデフォルトはユーザ定義の非 null 値になります。

マテリアライズされていないカラムを物理的に存在するカラムに変換することは、カラムの “インスタンス化” と呼ばれます。

マテリアライズされていないカラムの追加

alter table .. . not materialized を使用すると、マテリアライズされていないカラムを作成できます。

```
alter table table_name
add column_name datatype default constant_expression
not null [not materialized]
```

『リファレンス・マニュアル：コマンド』を参照してください。

注意　not materialized で null パラメータは指定できません。

たとえば、マテリアライズされていないカラム alt_title を titles テーブルに aaaaa のデフォルトを使用して追加するには、次のように入力します。

```
alter table titles
add alt_title varchar(24) default 'aaaaa'
not null not materialized
```

Adaptive Server によって、カラム *column_name* のデフォルトが指定された値を使用して作成され (存在しない場合)、そのデフォルトをカラムに関連付けて新しいカラムの syscolumns にエントリが挿入されます。

Adaptive Server では、テーブルの物理的データは変更されません。

マテリアライズされていないカラムを指定する alter table 句は、別のマテリアライズされていないカラムを作成する句または null 値の入力が可能な句と組み合わせることができます。マテリアライズされていないカラムを作成するための alter table は、完全なデータ・コピーを行う句 (カラムを削除するため、または null 値を取ることができないカラムを追加するための alter table など) とは組み合わせられません。

alter table 文で使用される場合、*constant_expression* は、定数 (たとえば 6 など) である必要があり、式は使用できません。*constant_expression* には、“6+4” などの式、関数 (getdate など)、または alter table コマンドで指定されているカラムに対して使用されているキーワード “user” を使用することはできません。

マテリアライズされていないカラムには、null ではないデフォルトの値が必要となります。デフォルト値を含める手順は次のとおりです。

- コマンド内でデフォルト値を明示的に指定する (たとえば `int default 0` など)
- バインドしているデフォルト値のあるユーザ定義のデータ型のある値を暗黙的に指定する。

注意　カラムの適正なデータ型に変換できない無効なデフォルト値を指定した場合、alter table はエラーを生成します。

column default cache size を使用して、カラムのデフォルト値のメモリ・プールを設定します。

マテリアライズされていないカラムが既に含まれたテーブル

Adaptive Server は、テーブルへの完全なデータ・コピーが必要な句があるコマンドを指定すると、すべてのマテリアライズされていないカラムをすぐにインスタンス化します。これには、reorg rebuild および null 値を取ることができないカラムを追加する alter table などのコマンドが含まれます。

マテリアライズされていないカラムの格納

マテリアライズされていないカラムがインスタンス化されると、Adaptive Server は、テーブル内のその他のロード同様の方法でそれらのカラムを格納します。

Adaptive Server が、固定長のマテリアライズされていないカラムをインスタンス化すると、ローは同等のマテリアライズされている固定長カラムの場合より大きな容量を占めます。また、インスタンス化されたマテリアライズされていないカラムには、同等の固定長カラムより大きな容量が必要です。データオンリー・ロック (DOL) テーブルには、カラムのオーバヘッドは 2 バイトです。全ページロック・テーブルでは、カラムのオーバヘッドは 1 バイト以上ですが、カラムの長さとそのロー内での物理的な配置によって変化します。

カラムは、マテリアライズされていないカラムを含むようにテーブルを変更したとき、そのテーブルにあったロー内でのみがマテリアライズされていません。ローを更新すると、マテリアライズされていないカラムは Adaptive Server によってインスタンス化されます。

マテリアライズされていないカラムには、null カラムまたは別のマテリアライズされていないカラムが続き、マテリアライズされたデータが続くことはありません。ロー内でカラムをインスタンス化すると、そのローより前に現れるマテリアライズされていないカラムは Adaptive Server によってインスタンス化されます。マテリアライズされていないカラムは、テーブルに追加された新しいローでインスタンス化されることもあります。

マテリアライズされていないカラムをインスタンス化すると、そのローのサイズが大きくなります。この現象が DOL テーブルで発生したときに含まれているページに十分な容量がない場合、Adaptive Server は拡大したローに対応するためにローを転送します (この操作は、null カラムを非 null 値で置き換える操作と同等の操作です)。

マテリアライズされていないカラムの変更

alter table ... replace を使用すると、マテリアライズされていないカラムのデフォルト値を変更できます。ただし、このコマンドは、以前のデフォルト値を使用しているカラムのデフォルト値は変更できません。alter table .... replace では、ユーザが追加または更新したカラムのデフォルト値を変更できます。

制限事項

Adaptive Server バージョン 15.7 では、既存のマテリアライズされているカラムをマテリアライズされていないカラムに変換するための alter table ... modify はサポートされていません。alter table ... modify をカラムに対して実行すると、そのテーブル内にあるマテリアライズされていないすべてのカラムがインスタンス化されます。

次の処理はできません。

- マテリアライズされていないカラムで bit、text、image、unitext、Java の各データ型を使用すること。
- マテリアライズされていないカラムは、IDENTITY カラムとして使用します。
- マテリアライズされていないカラムを追加するために alter table を使用するときに constraint 句、primary key 句、unique 句、または references 句を使用すること。
- マテリアライズされていないカラムを暗号化すること。
- マテリアライズされていない非 null カラムのあるデータを使用して Adaptive Server をバージョン 15.7 以前のバージョンにダウングレードすること。このようなダウングレードを行う場合は、まず、これらのカラムのあるテーブルで reorg rebuild を実行して、これらのカラムを標準の null 値を取れないカラムに変換する必要があります。

既存データの変更

テーブルの単独のロー、ローのグループ、またはすべてのローを変更するには、update コマンドを使用します。すべてのデータ修正文と同様、一度に 1 つのテーブルでだけデータを変更できます。

update は、変更する 1 つまたは複数のローと、新しいデータを指定します。新しいデータは、ユーザが指定する定数や式、または他のテーブルから取得するデータです。

update 文が整合性制約に違反すると、更新は実行されず、エラー・メッセージが生成されます。更新がキャンセルされるのは、たとえば、テーブルの IDENTITY カラムに影響する場合、追加される値のいずれかのデータ型が間違っている場合、関連するカラムやデータ型のいずれかに定義されているルールに違反する場合などです。

Adaptive Server では、1 つのローを複数回更新する update コマンドを発行できなくなることはありません。ただし、update の処理方法により、1 つの文からの複数回の更新は蓄積されません。つまり、update 文が同じローを 2 回修正する場合、2 回目の更新は、最初の更新からの新しい値ではなく、元の値に基づいて行われます。結果は、処理の順序に依存するので、予測できなくなります。

ビューの更新の制限については、「第 12 章 ビュー：データへのアクセスの制限」を参照してください。

注意 update コマンドはログに記録されます。text データ、unitext データ、image データの大きなブロックを変更している場合は、ログに記録されない writetext コマンドを使用してください。また、update 文あたりの上限は、およそ 125K です。「text データ、unitext データ、image データの変更」(235 ページ) にある writetext の説明を参照してください。

『リファレンス・マニュアル：コマンド』を参照してください。

update での *set* 句の使用

set 句はカラムおよび変更された値を指定します。where 句は、どのローが更新されるかを決定します。where 句がない場合、すべてのローの指定されたカラムは、set 句内で指定される値で更新されます。

注意 この項の例を実行する前に、pubs2 データベースの再インストールの方法を理解していることを確認してください。使用方法については、使用しているプラットフォームの『インストール・ガイド』および『設定ガイド』を参照してください。

たとえば publishers テーブルにあるすべての出版社がジョージア州のアトランタに移動した場合は、次のようにテーブルを更新します。

```
update publishers
set city = "Atlanta", state = "GA"
```

同様に、次の文を使用して、すべての出版社の名前を null に変更できます。

```
update publishers
set pub_name = null
```

更新には、計算されたカラムの値を使用できます。titles テーブルにある価格をすべて 2 倍にするには、次の文を使用します。

```
update titles
set price = price * 2
```

where 句がないため、この価格変更はテーブルのすべてのカラムに適用されます。

set 句での変数への代入

select 文で変数に代入するのと同じ方法で、update 文の set 句で変数に代入することができます。update で変数を使用すると、update 文とともに追加の select 文が使用される場合に発生するロックの競合と CPU の消費が減少されます。

次の例は、宣言された変数を使用して titles テーブルを更新します。

```
declare @price money
select @price = 0
update titles
    set total_sales = total_sales + 1,
    @price = price
    where title_id = "BU1032"
select @price, total_sales
    from titles
    where title_id = "BU1032"

                          total_sales
 ------------------------ -----------
                   19.99        4096

(1 row affected)
```

『リファレンス・マニュアル：コマンド』を参照してください。宣言された変数の詳細については、「ローカル変数」(475 ページ) を参照してください。

update での *where* 句の使用

where 句では、どのローが更新されるかを指定します。たとえば、著者名を Heather McBadden から Heather MacBadden に変更する場合は、次のようになります。

```
update authors
set au_lname = "MacBadden"
where au_lname = "McBadden"
and au_fname = "Heather"
```

update での *from* 句の使用

from 句を使用すると、1 つ以上のテーブルから更新中のテーブルにデータを抽出することができます。

たとえば、この章の前半に、au_id カラムに入力し、他のカラムにはダミー値や null 値を使用して、タイトルを持たない作家について titleauthor テーブルに新しいローを挿入する例がありました。このような作家である Dirk Stringer が『The Psychology of Computer Cooking』という本を著すと、titles テーブル内のこの作家の本にタイトル ID 番号が割り当てられます。タイトル ID 番号を追加することによって、titleauthor テーブルのこの作家のローを修正できます。

```
update titleauthor
set title_id = titles.title_id
from titleauthor, titles, authors
    where titles.title =
    "The Psychology of Computer Cooking"
    and authors.au_id = titleauthor.au_id
    and au_lname = "Stringer"
```

au_id ジョインを指定しない update は、titleauthor テーブルにある title_ids をすべて、『The Psychology of Computer Cooking』の ID 番号と同じになるように変更します。2 つのテーブルの構造がまったく同じであるが、一方には null フィールドがあり null 値がいくつか含まれているのに対し、もう一方は not null フィールドがない場合、select を使用して null テーブルから not null テーブルにデータを挿入することはできません。言い換えると、データのいずれかに null が含まれている場合、null の入力を許可しないフィールドは、null を許可するフィールドから選択することによって更新することはできません。

update 文の from 句の代替として、ANSI 規格に準拠するサブクエリを使用できます。

ジョインによる更新

次の例は、titles テーブルと publishers テーブルのカラムをジョインして、カリフォルニア州で出版されたすべての本の価格を 2 倍にしたものです。

```
update titles
  set price = price * 2
  from titles, publishers
  where titles.pub_id = publishers.pub_id
  and publishers.state = "CA"
```

IDENTITY カラムの更新

syb_identity キーワードを必要に応じてテーブル名で修飾して使用することにより、IDENTITY カラムを更新できます。たとえば、次の update 文は、IDENTITY カラムが 1 であるローを検出し、店の名前を “Barney’s” に変更します。

```
update stores_cal
set stor_name = "Barney’s"
where syb_identity = 1
```

text データ、*unitext* データ、*image* データの変更

長いテキスト値をデータベース・トランザクション・ログに格納したくない場合に、writetext を使用して、text、unitext 値、image 値を変更します。通常、update コマンドは常にログに記録されるため、update コマンドは使用しないでください。デフォルト・モードでは、writetext コマンドはログに記録されません。

注意　writetext をデフォルト (ログを取らない状態) で使用するには、システム管理者が sp_dboption を使用して select into/bulkcopy/pllsort を on に設定する必要があります。これにより、ログを取らないデータの挿入が許可されます。writetext を使用した後は、データベースをダンプする必要があります。データベースにログを取らない変更を行った後は、dump transaction を使用できません。

---

writetext コマンドは、影響するカラムのデータをすべて上書きします。カラムには、有効なテキスト・ポインタがあらかじめ含まれている必要があります。

次のように、textvalid() 関数を使用して、有効なポインタを調べます。

```
select textvalid("blurbs.copy", textptr(copy))
from blurbs
```

テキスト・ポインタを作成するには、次の 2 つの方法があります。

- insert を実行して text、unitext、または image カラムに実データを挿入する
- update を実行してカラムをデータまたは null で更新する

「初期化された」text カラムは、わずか数語を格納するのにも 2K の格納領域を使用します。insert によって明示的または暗黙的な null 値が text カラムに置かれると、Adaptive Server は、text カラムを初期化しないことによって、領域を節約します。次のコード例は、null テキスト・ポインタを使用して値を挿入し、テキスト・ポインタの存在を確認してから blurbs テーブルを更新します。テキストには説明コメントが埋め込まれています。

```
/* Insert a value with a text pointer.This could
** be done in a separate batch session.*/
```

```
insert blurbs (au_id) values ("267-41-2394")
/* Check for a valid pointer in an existing row.
** Use textvalid in a conditional clause; if no
** valid text pointer exists, update 'copy' to null
** to initialize the pointer.*/
if (select textvalid("blurbs.copy", textptr(copy))
    from blurbs
    where au_id = "267-41-2394") = 0
begin
    update blurbs
       set copy = NULL
      where au_id = "267-41-2394"
end
/*
** use writetext to insert the text into the
** column.The next statements put the text
** into the local variable @val, then writetext
** places the new text string into the row
** pointed to by @val.*/
declare @val varbinary(16)
select @val = textptr(copy)
    from blurbs
    where au_id = "267-41-2394"
writetext blurbs.copy @val
    "This book is a must for true data junkies."
```

この例で使用しているバッチ・ファイルとフロー制御言語の詳細については、「第 15 章 バッチおよびフロー制御言語の使用」を参照してください。

後続のゼロのトランケート

disable varbinary truncation 設定パラメータは、varbinary および binary の null データの後続のゼロを有効または無効にします(『システム管理ガイド 第 1 巻』の「第 5 章 設定パラメータ」を参照)。

デフォルトでは、サーバに対しては disable varbinary truncation はオフになっています。

Adaptive Server が後続のゼロをトランケートするように設定された場合、設定後に作成されるテーブルは、varbinary データを後続のゼロをトランケートしてから保存します。たとえば、disable varbinary truncation が 0 に設定された test1 を作成する場合は、次のようにします。

```
create table test1(col1 varbinary(5))
```

その後、後続のゼロがある varbinary データをいくつか挿入します。

```
insert into test1 values(0x12345600)
```

Adaptive Server によって、次のようにゼロがトランケートされます。

```
select * from test1
col1
------------
0x123456
```

ただし、テーブル test1 を削除および再作成して、disable varbinary truncation を 1 (オン) に設定して同じ手順を繰り返すと、Adaptive Server はゼロをトランケートしません。

```
select * from test1
col1
------------
0x12345600
```

Adaptive Server は、後続のゼロがあるデータとないデータは同等であると判断します (つまり、0x1234 は 0x123400 と同じであると見なされる)。

Adaptive Server は、disable varbinary truncation の現在の設定に応じてデータを保存するため、テーブルには、データ型が同じでも、後続のゼロがあるデータとないデータが混在している可能性があります。

- select into を実行してテーブル間でデータをコピーした場合、Adaptive Server は、格納されているとおりにデータをコピーします (つまり、disable varbinary truncation がオフになっている場合、後続のゼロがトランケートされています)。たとえば、上記の例で使用したテーブルで、disable varbinary truncation を無効にして、テーブル test1 のデータをテーブル test2 に選択します。

  ```
  sp_configure "disable varbinary truncation", 1

  select * into test2 from test1
  ```

  その後、同じデータをもう一度挿入します。

  ```
  insert into test2 select * from test1
  ```

  テーブル test2 は、後続のゼロをトランケートしません。これは、select into を disable varbinary truncation が 1 に設定された状態で実行したため、ターゲット・テーブルは、ソース・テーブルのプロパティを継承しないためです。ターゲット・テーブル内のデータは、select into が実行されたときに設定パラメータがどのように設定されていたかに応じて、トランケートまたはそのまま維持されます。

  ```
  select * from test2
  col1
  ------------
  0x12345600
  0x12345600
  ```

- バルク・コピー (bcp) によるデータの入力は、カラムの作成時に disable varbinary truncation がそのカラム上でどのように設定されていたかに応じて変化します。

- alter table は、特定のカラムのトランケートの動作を変更するためには使用できません。ただし、alter table を使用して追加するカラムは、disable varbinary truncation の値に応じてトランケートまたはそのまま維持されます。

  たとえば、テーブル test3 とカラム c1 を後続のゼロのトランケートが無効になった状態で作成した場合は次のようになります。

  ```
  sp_configure "disable varbinary truncation", 1
  create table test3(c1 varbinary(5))
  insert into test3 values(0x123400)
  ```

  c1 によって、後続のゼロが維持されます。

  ```
  select * from test3
  c1
  ----------
  0x123400
  ```

  ただし、後続のゼロのトランケートを有効にして alter table を新しいカラム c2 を追加するために使用した場合、次のようになります。

  ```
  sp_configure "disable varbinary truncation", 0
  alter table test3 add c2 varbinary(5) null
  insert into test3 values(0x123400, 0x123400)
  ```

  c2 は、後続のゼロを次のようにトランケートします。

  ```
  select * from test3
  c1           c2
  ----------   ----------
  0x123400           NULL
  0x123400         0x1234
  ```

後続のゼロは維持されます。

- ワークテーブル (disable varbinary truncation を 1 に設定した後)。次の最初の例には、後続のゼロが維持されるワークテーブルがあります。2 番目の例では、ワークテーブルは、最初の 6 桁のみを保存します。

  ```
  select 0x12345600 union select 0x123456
  -------------
  0x12345600


  select 0x123456 union select 0x12345600
  -------------
  0x123456
  ```

- 連結。次に例を示します。

  ```
  select 0x12345600 + col1, col1 from test1
                          col1
  --------------------      ------------
  0x123456001234560000      0x1234560000
  ```

```
0x1234560001234560        0x01234560
0x1234560012345600        0x12345600
0x123456000123456700      0x0123456700
```

- 関数。次に例を示します。

```
select bintostr(0x12340000)
------------
     1234000
```

- order by クエリと group by クエリ。次に例を示します。

```
select col1 from
(select 0x123456 col1 union all
select 0x12345600 col1) temp1 order by col1
col1
----------
0x123456
0x12345600
```

注意 クエリにワークテーブルがある場合、クエリの実行前に disable varbinary truncation 設定パラメータを無効にして Adaptive Server によるトランケートが行われないようにする必要があります。

- サブクエリ – 後続のゼロは、クエリにワークテーブルがない限り維持されます。クエリにワークテーブルがある場合、トランケートは disable varbinary truncation の値に応じて行われます。
- ダンプとロード – ダンプするテーブル・データに後続のゼロがある場合、ターゲット・データベース内の disable varbinary truncation の値にかかわりなくデータのロード時には後続のゼロは維持されます。
- 和集合 (上記のワークテーブルの例を参照)。
- convert。次に例を示します。

```
select convert(binary(5), 0x0000001000)
------------
0x0000001000
```

増分データ転送

transfer コマンドを使用すると、データを増分転送することができます。必要に応じて、異なる製品に増分転送することもできます。15.5 よりも前のバージョンの Adaptive Server では、1 台の Adaptive Server からもう 1 台の Adaptive Server にテーブル全体しか転送できませんでした。

注意 データ転送機能は、インメモリ・データベース・ライセンスを購入してインストールおよび登録した時点で、または RAP 製品をインストールした時点で、Adaptive Server によって有効化されます。

増分データ転送では、次のことが可能です。

- 増分転送用のマークが付けられた Adaptive Server のテーブルから、前回の転送後に変更されたデータだけをエクスポートすることができます。
- 通常のロックを取得しない、ローの取得順序を指定しない、読み取り中または更新中の他のデータを妨害しないで、テーブル・データを読み取ることができます。
- 選択したローは、定義されている受信側用にフォーマットした出力ファイルや名前付きパイプ (IQ (Sybase IQ)、ASE (Adaptive Server Enterprise)、バルク・コピー (bcp)、文字コード化された出力) に書き込むことができます。選択したすべてのローは暗号化せずに転送されます。また、暗号化されたカラムがローに含まれている場合は、デフォルトで復号化されてから転送されます。書き込むファイルは、Adaptive Server が稼働しているマシンで認識される必要があります (ファイルは、Adaptive Server がローカル・ファイルとして開くことのできる NFS ファイルにすることもできます)。
- 増分転送用テーブルの転送履歴が保持され、不要になったテーブルの転送履歴をユーザが削除できます。
- 一定の制限に従って、増分転送用として宣言されていないテーブルからデータをエクスポートします。
- 指定されたテーブルから、ロー単位でデータを転送します。現時点では、特定のカラムの選択、テーブル内の一部分の選択、SQL クエリ結果の転送は実行できません。

増分転送用テーブルのマーク付け

増分転送用として使用できることを示すマークをテーブルに付ける必要があります。システム・テーブルとワーク・テーブル以外のあらゆるテーブルを、増分転送用として指定できます。テーブルの作成時にこの指定を行うことも、または後で alter table を使用して指定することもできます。alter table を使用して、テーブルの増分転送用の指定を解除することもできます。

増分転送用のテーブルでは、次のことが行われます。

- 直前の転送後にローが変更された場合や、既存のローを変更するトランザクションや新しいローを挿入するトランザクションが転送の開始前に発生した場合は、ローが転送されます。

  これを行うにはローごとに追加の記憶域が必要です。追加の記憶域は、非表示の 8 バイト・カラムによってローに実装されます。

- 追加情報はテーブルの転送ごとに保存されます。この情報には、転送されたローのセットと番号、転送の開始時刻と終了時刻、転送用のデータ・フォーマット、および転送先ファイルのフル・パスを識別する情報が含まれます。

テーブルの増分転送用の指定を解除すると、増分転送をサポートするために各行に追加された変更、および保存されているテーブルの転送履歴が削除されます。

転送先ファイルからのテーブルの転送

transfer table コマンドを使用して、外部ファイルに含まれているテーブルから Adaptive Server にデータをロードします。『リファレンス・マニュアル：コマンド』を参照してください。

注意　テーブルをインポートするためには、Adaptive Server バージョン 15.5 で内部フォーマットを使用する必要があります。

テーブル内の既存データを変更したデータをロードする場合を除き、ロードするテーブルにユニークなプライマリ・インデックスは不要です (新しいデータは何の制約もなくロードできます)。ただし、ローがテーブル内に既存しているデータと重複し、データの重複を望まない場合は、データをロードするときに問題が発生します。この問題を避けるためには、ユニークなプライマリ・インデックスを使用することで、Adaptive Server が古いローを検索して新しいローと置き換えることができます。

ロードするテーブルにプライマリ・キーとしてユニークなインデックスが必要です (全ページのロック・テーブルのクラスタード・インデックス、またはデータのみのロック・テーブルの配置インデックス)。ユニークなインデックスを使うことで、transfer は重複するキーの挿入を検知して、内部 insert コマンドを update コマンドに変換することができます。このインデックスがないと、Adaptive Server は重複するプライマリ・キーを検知できません。更新されたローを挿入すると、次の問題が発生します。

- テーブルに他のユニークなインデックスがあり、挿入するローがそのインデックス内のキーと重複する場合、転送オペレーションの一部またはすべてが失敗する

- 挿入が成功しても、同じプライマリ・キーを持つ複数のローが誤ってテーブルに含まれる

次の例では、*/sybase/data* にある外部ファイル *titles.tmp* から Adaptive Server に pubs2.titles テーブルを転送します。

```
transfer table titles from '/sybase/data/titles.tmp' for ase
```

transfer table. . .to で使用するパラメータの一部は、transfer table...from に使用できません。ファイルからのデータのロードに不適切なパラメータがあると、エラーが発生して、transfer コマンドが停止します。Adaptive Server バージョン 15.5 には、from パラメータに使用するパラメータが含まれています。これらは今後のために予約されており、構文に含めた場合でも transfer はこれらのパラメータを無視します。from パラメータに使用できるパラメータは次のとおりです。

- column_order=*option* (for ase を使用したロードには適用されません。今後のために予約済み。)
- column_separator=*string* (for ase を使用したロードには適用されません。今後のために予約済み。)
- encryption={true | false} (for ase を使用したロードには適用されません。今後のために予約済み。)
- progress=*nnn*
- row_separator=*string* (for ase を使用したロードには適用されません。今後のために予約済み。)

Adaptive Server データ型から IQ への変換

表 7-5 では、Adaptive Server のデータ型が Sybase IQ でどのように表されるかを示します。Adaptive Server は、データ型を転送するときに必要な変換を適用して、指定された IQ 形式にデータを変換します。

表 7-5: Adaptive Server から IQ データ型への変換

Adaptive Server		IQ	
データ型	バイト・サイズ	データ型	バイト・サイズ
bigint unsigned bigint	8	bigint unsigned bigint	8
int unsigned int	4	int unsigned int	4
smallint	2	smallint	2
unsigned smallint	2	int	2
tinyint	1	tinyint	1
numeric(P,S) decimal(P,S)	2 ～ 17	numeric(P,S) decimal(P,S)	2 ～ 26
double precision	8	double	8

Adaptive Server		IQ	
real	4	real	4
float(P)	4, 8	float(P)	4, 8
money	8	money IQ はこれを numeric(19,4) で格納	16
smallmoney	4	smallmoney IQ はこれを numeric(10,4) で格納	8
bigdatetime datetime	8	datetime	8
smalldatetime	4	smalldatetime	8
date	4	date	4
time	4	time	8
bigtime	8	time	8
char(N)	1 ～ 16296	char(N)	1 ～ 16296
char(N) (null)	1 ～ 16296	char(N) (null)	1 ～ 16296
varchar(N) (null)	1 ～ 16296	varchar(N) (null)	1 ～ 16296
unichar(N)	1 ～ 8148	binary(N*2)	1 ～ 16296
unichar(N) null univarchar(N) (null)	1 ～ 8148	varbinary(N*2) (null)	1 ～ 16296
binary(N)	1 ～ 16296	binary(N)	1 ～ 16296
varbinary(N)	1 ～ 16296	varbinary(N)	1 ～ 16296
binary(N) null varbinary(N) (null)	1 ～ 16296	varbinary(N) (null)	1 ～ 16296
bit	1	bit	1
timestamp	8	varbinary(8) null	8

Adaptive Server のデータ型を IQ のデータ型に変換するときは、次のことを考慮してください。

- IQ と Adaptive Server で同じ精度と位取りを定義します。
- float の記憶サイズは、精度に応じて 4 バイトまたは 8 バイトです。精度の値を指定しないと、Adaptive Server は float を double precision として格納しますが、IQ は real として格納します。Adaptive Server は、IQ に転送するために浮動小数点データを他のフォーマットに変換しません。概数値データ型を使用する必要がある場合は、float としてではなく double または real として指定します。
- Adaptive Server では char、unichar、binary のデータ型を持つカラムの最大長はインストールのページ・サイズで決まります。表 7-5 に指定される最大サイズは、16K ページで利用可能な最大カラムです。

- 通常 Adaptive Server では、Unicode 文字を格納するために 1 文字あたり 2 バイトが必要です。IQ には Unicode データ型が含まれないため、Adaptive Server は unichar(N) を binary(N X 2) として IQ に転送します。ただし、Adaptive Server では Unicode 文字を変換するわけではありません。Adaptive Server は Unicode 文字列に NULL(0x00) を埋め込んで IQ に転送します。
- IQ にはネイティブ Unicode データ型がありません。Adaptive Server は、バイナリ・データとして Unicode 文字列を IQ に転送します。つまり、Unicode 1 文字ごとに 2 バイト長のデータが転送されます。たとえば、Adaptive Server の unichar(40) は IQ では binary(80) に変換されます。IQ は、転送後の Unicode データを文字列として表示できません。

転送情報の保管

転送情報は、次のように格納されます。

- spt_TableTransfer － テーブル転送の結果の保管と取得は spt_TableTransfer テーブルで行われます。
- monTableTransfer － テーブルの転送履歴情報が含まれています。この情報には、現在進行中の転送と完了済みの転送の情報が含まれます。

spt_TableTransfer

transfer table コマンドで指定したテーブルから取得される正常な転送の結果は、以降の転送のデフォルトとして使用されます。たとえば、次のコマンドを出すとします (ローとカラムのセパレータを含む)。

```
transfer table mytable for csv
```

テーブル mytable を次に転送するとき、transfer コマンドは for csv、同じロー・セパレータ、同じカラム・セパレータをデフォルトで使用します。

各データベースには、独自のバージョンの spt_TableTransfer があります。このテーブルには、同じデータベース内で増分転送のマークが付けられているテーブルのテーブル転送履歴だけが保管されます。

max transfer history 設定パラメータは、Adaptive Server が各データベースの spt_TableTransfer テーブルに保持する転送履歴の数を制御します。『システム管理ガイド 第 1 巻』の「第 5 章 設定パラメータ」を参照してください。

データベース所有者は、sp_setup_table_transfer を使用して spt_TableTransfer テーブルを作成します。sp_setup_table_transfer にはパラメータがなく、現在のデータベースで動作します。

spt_TableTransfer には、成功した転送と失敗した転送の両方に関する履歴情報が保管されます。処理中の転送に関する情報は保管されません。

spt_TableTransfer はシステム・テーブルではなくユーザ・テーブルです。このテーブルは Adaptive Server の作成時には作成されませんが、sp_setup_transfer_table を使用して手動で作成しないと、転送で使用できるテーブルを持つあらゆるデータベース内に Adaptive Server によって自動的に作成されます (spt_TableTransfer テーブルを手動で作成すると、Adaptive Server がこのテーブルを自動作成するときに発生する可能性のある予期しないエラーを防ぐことができる場合があります)。

sp_help は増分転送をテーブル属性として報告します。

次に示すのは spt_TableTransfer のカラムです。

カラム	データ型	説明
end_code	unsigned smallint not null	転送の終了ステータス 0 － 成功 エラー・コード － 失敗
id	int not null	転送されるテーブルのオブジェクト ID。
ts_floor	bigint not null	開始トランザクションのタイムスタンプ。
ts_ceiling	bigint not null	ローがコミットされないため転送されなかった時点の直前のトランザクションのタイムスタンプ。
time_begin	datetime not null	• 転送の開始日時 • transfer が実装前に失敗した場合は、Adaptive Server がコマンドの設定を開始した時刻。それ以外の場合は、コマンドが最初のデータを出力ファイルに送信した時刻。
time_end	datetime not null	• 転送の終了日時 • transfer コマンドが失敗した場合は、失敗した時刻。それ以外の場合は、コマンドがデータを送信してファイルを閉じた時刻。
row_count	bigint not null	転送されたローの数。
byte_count	bigint not null	書き込まれたバイト数。
sequence_id	int not null	テーブルの転送ごとにユニークな、転送の追跡番号。
col_order	tinyint not null	出力のカラム順序を表す番号。 • 1 － id • 2 － offset • 3 － name • 4 － name_utf8

カラム	データ型	説明
output_to	tinyint not null	出力フォーマットを表す番号。 • 1 − ase • 2 − bcp • 3 − csv • 4 − iq
tracking_id	int null	カスタマ定義の追跡 ID (オプション)。tracking_id = nnn で使用しない場合、このカラムは null になります。
pathname	varchar(512) null	出力ファイル名。
row_sep	varchar(64) null	for csv で使用されるロー・セパレータ。
col_sep	varchar(64) null	for csv で使用されるカラム・セパレータ。

monTableTransfer

monTableTransfer テーブルは、次の情報を示します。

- Adaptive Server が現在メモリ内に情報を保持しているテーブルの転送履歴情報。現在のすべてのテーブル情報を十分に保持できるだけの大きさに Adaptive Server のメモリを設定していない場合を除き、Adaptive Server が前回再起動されてからアクセスされたテーブルが該当します。
- 現在進行中の転送に関する情報、Adaptive Server が情報をメモリに保持しているテーブルについて完了した転送の情報。これには、次のテーブルの情報が含まれます。
  - 増分転送のマークがあること
  - Adaptive Server を再起動してから少なくとも 1 度は転送で使用されたこと
  - 他のテーブルで使用されていない記述があること (monTableTransfer は、これまで転送されたすべてのテーブルを検索する場合、すべてのデータベースを検索するわけではなく、最近の転送を行ったアクティブなテーブルのセットのみを検索します)。

『リファレンス・マニュアル：テーブル』の「monTableTransfer」を参照してください。

例外とエラー

データ転送ユーティリティは、次のようなデータ転送には使用できません。

- パイプ上で for bcp フォーマットを使用したデータ転送
- Windows プラットフォーム上のパイプの使用

Adaptive Server は、次の場合にエラー・メッセージを生成します。

- 存在しないテーブルを転送しようとした。

- テーブルではないオブジェクトを転送しようとした。
- 自分が所有していないテーブル、転送のパーミッションを付与されていないテーブル、sa_role 権限のないテーブルを転送しようとした。
- カラムを復号化する特定のパーミッションがないのに、暗号化されたカラムを含むテーブルから転送中にカラムを復号化しようとした。
- テーブルに text カラムまたは image カラムが含まれている場合に for iq を転送しようとした。
- offset ではなくカラムの順序を指定して for ase を転送しようとした。
- id ではなくカラムの順序を指定して for bcp を転送しようとした。
- システム・カタログを指名する alter table...set transfer table on コマンドを出そうとした。

転送の障害の他の理由として、次のことが考えられます。

- 要求したファイルを閉じることができない。
- ローに含まれていないカラム (ローに含まれない状態で格納されている text、unitext、image、Java カラム) を持つテーブルは転送できない。
- ファイルを開けない。ディレクトリが存在しており、Adaptive Server にディレクトリへの write パーミッションがあることを確認してください。
- 転送のテーブルごとに保存されたデータを保管しておく十分なメモリを Adaptive Server で取得できない。このような場合は、Adaptive Server で使用できるメモリ量を増やしてください。

増分データ転送のサンプル・セッション

このチュートリアルでは、データを外部ファイルに転送する方法、テーブル内のデータを変更する方法、次に、もう一度 transfer コマンドを使用してこの外部ファイルからテーブルを再移植する方法を説明します。また、transfer コマンドでは、データはファイルに追加されること、つまり、データの上書きは行わないことを示します。

注意　この例では、データの転送元と転送先は同じテーブルになっていますが、通常のユーザ・シナリオでは、データの転送元と転送先は異なるテーブルとなります。

1　転送履歴を格納する spt_TableTransfer テーブルが作成されます。

```
sp_setup_table_transfer
```

2 max transfer history を設定します。デフォルトは 10 です。つまり、Adaptive Server は増分転送用のマークの付いたテーブルごとに、成功した転送と失敗した転送を 10 件ずつ保持します。次の例では、max transfer history の値を 10 から 5 に変更します。

```
sp_configure 'max transfer history', 5

Parameter Name               Default     Memory Used
Config Value    Run Value              Unit
Type                       Instance Name
-------------------------     -------     -----------
--------------   --------------------   -------
---------------          ---------------
max transfer history      10              0
5               5                        bytes
dynamic                  NULL
```

3 transfer 属性が有効でデータロー・ロックを使用する transfer_example テーブルを作成します。

```
create table transfer_example (
f1 int,
f2 varchar(30),
f3 bigdatetime,
primary key (f1)
) lock datarows
   with transfer table on
```

4 transfer_example テーブルにサンプル・データを移植します。

```
set nocount on
declare @i int, @vc varchar(1024), @bdt bigdatetime
select @i = 1
while @i <= 10
begin
   select @vc = replicate(char(64 + @i), 3 * @i)
   select @bdt = current_bigdatetime()
   insert into transfer_example values ( @i, @vc, @bdt )
   select @i = @i + 1
end
set nocount off
```

このスクリプトで次のデータが生成されます。

```
select * from transfer_example
order by f1
f1       f2                              f3
--------- ------------------------------ ----------------------------
   1      AAA                             Jul 17 2009  4:40:14.465789PM
   2      BBBBBB                          Jul 17 2009  4:40:14.488003PM
   3      CCCCCCCCC                       Jul 17 2009  4:40:14.511749PM
   4      DDDDDDDDDDDD                    Jul 17 2009  4:40:14.536653PM
```

```
   5      EEEEEEEEEEEEEE                Jul 17 2009  4:40:14.559480PM
   6      FFFFFFFFFFFFFFFFF             Jul 17 2009  4:40:14.583400PM
   7      GGGGGGGGGGGGGGGGGGGG          Jul 17 2009  4:40:14.607196PM
   8      HHHHHHHHHHHHHHHHHHHHHHH       Jul 17 2009 4:40:14.632152PM
   9      IIIIIIIIIIIIIIIIIIIIIIII   Jul 17 2009  4:40:14.655184PM
  10      JJJJJJJJJJJJJJJJJJJJJJJJJJJ  Jul 17 2009  4:40:14.678938PM
```

5　for ase フォーマットを使用して transfer_example データを外部ファイルに転送します。

```
transfer table transfer_example
to 'transfer_example-data.ase'
for ase

(10 rows affected)
```

このデータ転送で次の履歴レコードが spt_TableTransfer に作成されます。

```
select id, sequence_id, end_code, ts_floor, ts_ceiling, row_count
from spt_TableTransfer
where id = object_id('transfer_example')
id        sequence_id      end_code     ts_floor       ts_ceiling
row_count
---------   -----------       --------      -------------   -------------------
592002109  1               0           0              5309                  10
```

6　transfer_example の transfer 属性を無効にして、増分データを受信するために受信テーブルで transfer 属性を有効にする必要がないことを示します (alter table を実行するためには、データベースで select into が有効になっていることが必要です)。

```
alter table transfer_example
set transfer table off
```

alter table コマンドが実行されると、spt_TableTransfer が空になります。

```
select id, sequence_id, end_code, ts_floor, ts_ceiling, row_count
from spt_TableTransfer
where id = object_id('transfer_example')
id        sequence_id      end_code     ts_floor       ts_ceiling
row_count
---------   -----------       --------      -------------   -------------------

(0 rows affected
```

7　transfer_example を更新して文字データを no data に設定し、テーブルに元のデータが含まれていないことを確認できるように bigdatetime カラムに日時を指定します。

```
update transfer_example
set f2 = 'no data',
f3 = 'Jan 1, 1900 12:00:00.000001AM'

(10 rows affected)
```

update を実行すると、transfer_example には次のデータが含まれます。

```
select * from transfer_example
order by f1
f1          f2                              f3
----------- ------------------------------- --------------------------
1           no data                         Jan  1 1900 12:00:00.000001AM
2           no data                         Jan  1 1900 12:00:00.000001AM
3           no data                         Jan  1 1900 12:00:00.000001AM
4           no data                         Jan  1 1900 12:00:00.000001AM
5           no data                         Jan  1 1900 12:00:00.000001AM
6           no data                         Jan  1 1900 12:00:00.000001AM
7           no data                         Jan  1 1900 12:00:00.000001AM
8           no data                         Jan  1 1900 12:00:00.000001AM
9           no data                         Jan  1 1900 12:00:00.000001AM
10          no data                         Jan  1 1900 12:00:00.000001AM

(10 rows affected)
```

8 サンプル・データを外部ファイルから transfer_example に転送します。transfer_example には増分転送用のマークが付いていませんが、データはテーブルに転送できます。transfer_example にはユニークなプライマリ・インデックスがあるため、送られてくるローが既存のデータを置き換え、duplicate key エラーは発生しません。

```
transfer table transfer_example
from 'transfer_example-data.ase'
for ase

(10 rows affected)
```

9 transfer_example のすべてのデータを選択して、受信データによって変更されたデータが置き換えられたことを確認します。transfer により、transfer_example.f2 テーブルと transfer_example.f3 テーブルの内容は、*transfer_example-data.ase* 出力ファイルに格納されていた、最初にテーブル用に作成されたデータで置換されました。

```
select * from transfer_example
order by f1
f1      f2                              f3
------- ------------------------------- --------------------------
   1    AAA                             Jul 17 2009 4:40:14.465789PM
   2    BBBBBB                          Jul 17 2009  4:40:14.488003PM
   3    CCCCCCCCC                       Jul 17 2009  4:40:14.511749PM
   4    DDDDDDDDDDDD                    Jul 17 2009  4:40:14.536653PM
   5    EEEEEEEEEEEEEE                  Jul 17 2009  4:40:14.559480PM
   6    FFFFFFFFFFFFFFFFF               Jul 17 2009  4:40:14.583400PM
   7    GGGGGGGGGGGGGGGGGGGG            Jul 17 2009  4:40:14.607196PM
   8    HHHHHHHHHHHHHHHHHHHHHHHH        Jul 17 2009  4:40:14.632152PM
   9    IIIIIIIIIIIIIIIIIIIIIIIII       Jul 17 2009  4:40:14.655184PM
  10    JJJJJJJJJJJJJJJJJJJJJJJJJJJ  Jul 17 2009  4:40:14.678938PM
```

10　後続の転送が前のパラメータをデフォルトで使用するように、transfer_example に対して transfer をもう一度有効にします。

```
alter table transfer_example
set transfer table on
(10 rows affected)
```

新しいローでのデータの置き換え

一部のローのキー値を変更した場合に増分転送を実行すると、そのテーブルに対する次の transfer は、変更されたキー・データのローを新しいデータとみなします。ただし、キーの変更されていないローのデータが置き換えられます。

1　transfer_example では f1 カラムをプライマリ・キー・カラムとして使用します。Adaptive Server はこのカラムを使用して、送られてきたローに新しいデータが含まれているかどうか、既存のローを置き換えるかどうかを判断します。

次の例では、キー 3、5、7 にそれぞれ 10 を追加して、これらのキーを持つローを置き換えます。

```
update transfer_example
set f1 = f1 + 10
where f1 in (3,5,7)
(3 rows affected)
```

これで、transfer_example にはキー 13、15、および 17 を持つローが入りました。transfer は、これらのローを新しいローとみなします。同じデータを transfer_example に転送すると、transfer はキー 3、5、および 7 を持つローを挿入して、キー 13、15、および 17 を持つローを保持します。

```
transfer table transfer_example
from 'transfer_example-data.ase'
for ase
(10 rows affected)
```

2　f2 と f3 について、ロー 3 のデータはロー 13 と、ロー 5 はロー 15 と、ロー 7 はロー 17 と同じであることを確認してください。

```
select * from transfer_example
order by f1
f1     f2                              f3
------- ------------------------------ ----------------------------
1       AAA                             Jul 17 2009 4:40:14.465789PM
2       BBBBBB                          Jul 17 2009  4:40:14.488003PM
3       CCCCCCCCC                       Jul 17 2009  4:40:14.511749PM
4       DDDDDDDDDDDD                    Jul 17 2009  4:40:14.536653PM
5       EEEEEEEEEEEEEE                  Jul 17 2009  4:40:14.559480PM
6       FFFFFFFFFFFFFFFFF               Jul 17 2009  4:40:14.583400PM
7       GGGGGGGGGGGGGGGGGGGG            Jul 17 2009  4:40:14.607196PM
8       HHHHHHHHHHHHHHHHHHHHHHHH        Jul 17 2009  4:40:14.632152PM
9       IIIIIIIIIIIIIIIIIIIIIIIII       Jul 17 2009  4:40:14.655184PM
```

```
10      JJJJJJJJJJJJJJJJJJJJJJJJJJJ  Jul 17 2009  4:40:14.678938PM
13      CCCCCCCCC                    Jul 17 2009  4:40:14.511749PM
15      EEEEEEEEEEEEEEE              Jul 17 2009  4:40:14.559480PM
17      GGGGGGGGGGGGGGGGGGGG         Jul 17 2009  4:40:14.607196PM

(13 rows affected)
```

3 transfer_example をもう一度転送すると、13 個のすべてのローが転送されます。キー 3、5、および 7 で転送したローは既存のローを置き換えたため、Adaptive Server はこれらを新しいローとみなします (この例では追跡 ID 値 101 を使用します)。

```
transfer table transfer_example
to 'transfer_example-data-01.ase'
for ase
with tracking_id = 101
(13 rows affected)
```

4 ローを変更して、増分 transfer では直前の転送後に変更されたローだけが転送されることを示します (この更新が影響するのは 3 つのロー)。

```
update transfer_example
set f3 = current_bigdatetime()
where f1 > 10
(3 rows affected)
```

5 このテーブルをもう一度転送して、変更された 3 つのローだけが転送されたことを確認します。for ase を指定する必要はありません。Adaptive Server は、前の転送で使用されたパラメータをデフォルトで使用します。

```
transfer table transfer_example
to 'transfer_example-data-02.ase'
with tracking_id = 102
(3 rows affected)
```

6 手順 3 の tracking_id を使用して、転送情報を表示します。

```
select id, sequence_id, end_code, ts_floor, ts_ceiling, row_count
from spt_TableTransfer
where id = object_id('transfer_example')
and tracking_id = 101
id          sequence_id   end_code    ts_floor             ts_ceiling
row_count
-----------   -----------    --------     --------------------   --------------
------
592002109    3             0           5309                 5716
13
```

データの削除

delete は、単一ローのオペレーションでも、複数のローのオペレーションでも機能します。

where 句はどのローが削除されるかを指定します。delete 文内に where 句が指定されていないと、テーブルのすべてのローが削除されます。『リファレンス・マニュアル：コマンド』を参照してください。

delete での *from* 句の使用

delete キーワードの直後のオプションの from は、他のバージョンの SQL との互換性のために含まれています。delete 文の 2 番目の位置の from 句は Transact-SQL で、1 つまたは複数のテーブルからのデータの選択および最初に指定されたテーブルからの対応データの削除を可能にします。from 句で選択したローは、delete コマンドの条件を指定します。

複雑な企業間の取引の結果、Oakland のすべての作家とその本が別の出版社によって買収されたとします。これらすべての本を titles テーブルからただちに削除する必要がありますが、そのタイトルまたは ID 番号がわかりません。わかっているのは、作家の名前と住所だけです。

authors テーブル内の都市に Oakland を持つローの作家 ID 番号を見つけ、その番号を使用して titleauthor テーブル内の本のタイトル ID 番号を検索することによって、titles のローを削除できます。つまり、titles テーブルで削除するローを検索するには、3 段階のジョインが必要です。

3 つのテーブルはすべて delete 文の from 句に含まれています。しかし削除されるのは、where 句の条件を満たす titles テーブルのローだけです。titles 以外のテーブルにある、関連するローを削除するには、別の delete 文を使用する必要があります。

これが正しい文になります。

```
delete titles
from authors, titles, titleauthor
where titles.title_id = titleauthor.title_id
and authors.au_id = titleauthor.au_id
and city = "Oakland"
```

pubs2 データベースの deltitle トリガにより、ユーザは実際にはこの削除を実行できません。これは、このトリガが、sales テーブルに記録されている売り上げを持つタイトルの削除を許可しないためです。

IDENTITY カラムからの削除

IDENTITY カラムを含むテーブルの delete 文で syb_identity キーワードを使用できます。たとえば、次の文は、row_id が 1 であるローを削除します。

```
delete sales_monthly
where syb_identity = 1
```

IDENTITY カラムのローを削除した後で、テーブルの IDENTITY カラムの番号付けシーケンスにあるギャップを削除することが必要になる場合があります。「bcp を使用してテーブルの IDENTITY カラムの番号を付け直す」(224 ページ)を参照してください。

テーブルからのすべてのローの削除

テーブル内のすべてのローを削除するには、truncate table を使用します。truncate table は、ほとんどの場合において、条件のない delete 文よりも高速です。それは、truncate table がデータ・ページ全体の割り付け解除をログに記録するだけであるのに対し、delete はそれぞれの変更をログに記録するためです。truncate table は、テーブルのデータとインデックスが占有していた領域をただちに解放します。解放された領域は、任意のオブジェクトが使用できます。すべてのインデックスのディストリビューション・ページも割り付けを解除されます。テーブルに新しいローを追加した後に、update statistics を実行してください。

delete と同様に、truncate table によって空になったテーブルは、drop table コマンドを入力しないかぎり、インデックスおよびその他の関連オブジェクトとともに、データベースに残ります。

別のテーブルに、参照整合性制約を介してそのテーブルを参照するローがある場合は、truncate table を使用できません。外部テーブルからローを削除するか、外部テーブルをトランケートしてから、プライマリ・テーブルをトランケートします。「参照整合性制約を作成するための一般的な規則」(290 ページ) を参照してください。

truncate table 構文

truncate table の構文は次のとおりです。

truncate table [[*database*.]*owner*.]*table_name*
[partition *partition_name*]

たとえば、sales のデータをすべて削除するには、次のように入力します。

```
truncate table sales
```

partition 句については、「第 10 章 テーブルとインデックスの分割」を参照してください。

truncate table を使用するパーミッションは、drop table と同様に、デフォルトではテーブル所有者にあり、譲渡はできません。

truncate table コマンドは delete トリガの対象にはなりません。「第 20 章 トリガ：参照整合性」を参照してください。

第 8 章

データベースおよびテーブルの作成

データベースとテーブル

データベースは、テーブルなどの、互いに関係するデータベース・オブジェクトのセットに情報 (データ) を格納します。「テーブル」は、個々のデータ項目が含まれる関連付けられたカラムを持つローの集合です。データベースおよびテーブルを作成するときに、データをどのように構成するかを決定します。この処理は「データ定義」と呼ばれます。

Adaptive Server データベース・オブジェクトには、次のものが含まれます。

- テーブル
- ルール
- デフォルト
- ストアド・プロシージャ
- トリガ
- ビュー

- 参照整合性制約
- 検査整合性制約
- 関数
- 計算カラム
- 分割条件

カラムと「データ型」は、テーブルに含まれているデータの型を定義します。これについては、この章で説明します。インデックスには、データがテーブル内でどのように構成されるかが記述されています。これらは Adaptive Server によってデータベース・オブジェクトとはみなされておらず、sysobjects にはリストされません。インデックスについては、「第 13 章 テーブルのインデックスの作成」を参照してください。

データベース内のデータ整合性の保持

「データの整合性」とは、あるデータベース内でのデータの正当性および完全性を意味します。データの整合性を保つには、ユーザがデータベース内で挿入、削除、または更新できるデータ値を制約または制限します。たとえば、pubs2 と pubs3 データベースのデータ整合性によって、titles テーブルにある本のタイトルには、publishers テーブルに出版社があることが必要です。有効な出版社がない本を titles に挿入することはできません。これは pubs2 または pubs3 のデータ整合性に違反するためです。

Transact-SQL は、ルール、デフォルト、インデックス、トリガなど、データベース内の整合性を保つためのメカニズムをいくつか提供しています。このメカニズムによって、次の種類のデータ整合性を維持できます。

- 「必要条件整合性」– テーブル・カラムでは、どのローにも、有効な値が 1 つ入っている必要があります。null 値は許可されません。create table 文では、カラムの null 値を制限できます。
- 「検査整合性」または「有効性整合性」– テーブル・カラムに挿入されるデータ値を限定、制限します。トリガやルールを使用して、この種類のデータ整合性を保つことができます。
- 「一意性整合性」– 1 つ以上のテーブル・カラムに対して null 以外の同じ値を持つテーブル・ローが 2 つとありません。インデックスを使用してこのデータ整合性を保つことができます。
- 「参照整合性」– テーブル・カラムに挿入されたデータは、対応するデータを、別のテーブルのカラムまたは同じテーブルの別のカラムにあらかじめ持っている必要があります。1 つのテーブルは、最大で 192 の参照を持つことができます。

データベース内のデータ値の一貫性も、データ整合性の例です。これについては、「第 23 章 トランザクション：データの一貫性およびリカバリ」を参照してください。

Transact-SQL では、ルール、デフォルト、インデックス、およびトリガを使用する代わりに、SQL 規格で定義されたデータ整合性を保つために、create table 文の一部として「整合性制約」を提供しています。これらの整合性制約については、このマニュアルの後半で説明します。

データベース内のパーミッション

データベースとデータベース・オブジェクトの作成および削除を実行できるかどうかは、ユーザのパーミッションまたは権限によって異なります。通常は、ユーザが行う作業の種類や、ユーザに必要な機能に基づいて、システム管理者またはデータベース所有者がユーザのパーミッションを設定します。これらのパーミッションは、インストールやデータベース内のユーザごとに異なるものを設定できます。

自分に付与されているパーミッションを調べるには、次のコマンドを実行します。

```
sp_helprotect user_name
```

ただし、*user_name* は、実行するユーザの Adaptive Server ログイン名です。

pubs2 データベースおよび pubs3 データベースには、それぞれの sysusers システム・テーブル内に **guest** ユーザ名があります。pubs2 と pubs3 を作成するスクリプトは、“guest” にさまざまなパーミッションを付与します。

“guest” メカニズムは、Adaptive Server 上に「**login**」を持つユーザ、つまり、master..syslogins にリストされているユーザであれば誰でも、pubs2 と pub3 にアクセスでき、テーブル、インデックス、デフォルト、ルール、プロシージャなどのオブジェクトの作成と削除のパーミッションを持つことを意味します。“guest” ユーザ名を使用すると、一定のストアド・プロシージャの使用、ユーザ定義データ型の作成、データベースへの問い合わせ、データベースのデータの修正も可能になります。

pubs2 または pubs3 データベースを使用するには、use コマンドを実行します。Adaptive Server は、ユーザが pubs2..sysusers または pubs3..sysusers にユーザ自身の名前でリストされているかどうかをチェックします。リストされていない場合、ユーザはアクションを起こさなくても guest として認識されます。ユーザが pubs2 または pubs3 の sysusers テーブルにリストされている場合、Adaptive Server は、ユーザをそのユーザ自身として認識し、“guest” のパーミッションとは異なるパーミッションを付与します。

注意　この章のすべての例では、ユーザを “guest” として扱っています。

ほとんどのユーザは、“guest” メカニズムを使用して、master データベースにあるシステム・テーブルを参照できます。master データベース内でユーザ名が認識されないユーザも、“guest” という名前のユーザとして許可され、取り扱われます。“guest” ユーザは、インストール時に master データベースを作成するスクリプトで master データベースに追加されます。

データベース所有者である “dbo” は、sp_adduser を使用して、任意のユーザ・データベースに “guest” ユーザを追加できます。システム管理者は、使用するデータベースで自動的にデータベース所有者となります。『システム管理ガイド 第 1 巻』の「第 13 章 Adaptive Server のセキュリティ管理について」を参照してください。

データベースの使用と作成

データベースは、関係テーブルと、ビュー、インデックスなどその他のデータベース・オブジェクトの集合です。

Adaptive Server をインストールすると、次の「システム・データベース」が含まれています。

- master − ユーザ・データベースおよび Adaptive Server の動作を総括して制御します。
- sybsystemprocs − システム・ストアド・プロシージャが含まれます。
- sybsystemdb − 分散トランザクションに関する情報が含まれます。
- tempdb − テンポラリ・オブジェクトが格納されます。これには、“tempdb..” というプレフィクスの付いた名前で作成されるテンポラリ・テーブルが含まれます。
- model − Adaptive Server によって新しいユーザ・データベースを作成するためのテンプレートとして使用されます。

さらに、システム管理者は、次に示すオプションのデータベースをインストールできます。

- pubs2 − 出版物に関する操作を表すデータを含むサンプル・データベースです。このデータベースを使用して、サーバ接続をテストしたり Transact-SQL について学習したりできます。Adaptive Server のマニュアルに掲載されている例のほとんどでは、pubs2 データベースを使用します。
- pubs3 − pubs2 のバージョンの 1 つで、参照整合性の例を使用します。pubs3 には、自己参照カラムを使用する store_employees テーブルがあります。また、pubs3 は sales テーブルに IDENTITY カラムが含まれます。さらに、その pubs3 マスタ・テーブルのプライマリ・キーはノンクラスタード・ユニーク・インデックスを使用しており、titles テーブルには numeric データ型の例があります。
- interpubs − pubs2 に類似したデータベースで、フランス語とドイツ語のデータが入っています。
- jpubs − pubs2 と同じようなデータベースで、日本語のデータが入っています。日本語モジュールをインストールした場合はこれを使用してください。

これらのオプションのデータベースはユーザ・データベースです。データはすべてユーザ・データベースに格納されます。Adaptive Server は、各データベースをシステム・テーブルによって管理します。master データベースおよびその他のデータベースにある「データ辞書」テーブルは、システム・テーブルとみなされます。

データベースの選択：*use*

use コマンドを使用すると、認識されているユーザに限り、次のようにして既存のデータベースにアクセスできます。

```
use database_name
```

たとえば、pubs2 データベースにアクセスするには、次のように入力します。

```
use pubs2
```

Adaptive Server にログインすると自動的に master データベースに接続される可能性があるため、別のデータベースを使用する場合は use コマンドを発行してください。ユーザまたはシステム管理者は、sp_modifylogin を使用して、最初に接続するデータベースを変更できます。別のユーザのデフォルト・データベースを変更できるのは、システム管理者だけです。

create database によるユーザ・データベースの作成

システム管理者によって create database コマンドを使用するパーミッションを付与されていれば、新しいデータベースを作成できます。新しいデータベースを作成するときは、master データベースを使用してください。多くの企業では、データベースはすべてシステム管理者が作成します。データベースの作成者はその所有者になります。別のユーザから作成したデータベースの所有権の譲渡を受けるには、sp_changedbowner を使用できます。

データベース所有者は、ユーザへのデータベース・アクセスの付与、およびユーザのその他の一定のパーミッションの付与と取り消しに責任があります。編成によっては、データベース所有者が、データベースの定期バックアップの管理、およびシステム障害の場合の再ロードにも責任がある場合があります。データベース所有者は、setuser コマンドを使用して、データベースに対する別の任意のユーザのパーミッションを一時的に取得することができます。

各データベースには、わずかなデータしか含まれていない場合でも莫大な領域が割り付けられているため、create database コマンドを使用するパーミッションが付与されない場合があります。

create database コマンドの最も簡単な形式は次のとおりです。

```
create database database_name
```

newpubs データベースという新しいデータベースを作成するには、自分が pubs2 以外の master データベースを使用していることを確認してから、次のコマンドを入力します。

```
use master
create database newpubs
drop database newpubs
use pubs2
```

データベース名は、Adaptive Server 上でユニークであり、「識別子」(10 ページ)で説明されている識別子の規則に従っている必要があります。Adaptive Server は最大 32,767 個のデータベースを管理できます。一度に作成できるデータベースの数は 1 つだけです。データベースのセグメント (1 つまたは複数のデータベース・デバイスを指すラベル) の最大数は 32 です。

Adaptive Server は、model データベースのコピーとして新しいデータベースを作成します。model データベースには、どのユーザ・データベースにも属するシステム・テーブルが含まれます。

新しいデータベースの作成は、master データベース・テーブルの sysdatabases と sysusages に記録されます。

『リファレンス・マニュアル：コマンド』および『リファレンス・マニュアル：テーブル』を参照してください。

この章では、with override を除くすべての create database オプションについて説明します。with override の詳細については、『システム管理ガイド 第 2 巻』の「第 6 章 ユーザ・データベースの作成と管理」を参照してください。

on 句

オプションの on 句を使用すると、データベースの格納場所と、それに割り付ける領域の大きさ (メガバイト) を指定できます。キーワード default を使用すると、データベースは、master データベース・テーブルの sysdevices に示される、デフォルト・データベース・デバイスのプール内の使用可能なデータベース・デバイスに割り当てられます。どのデバイスがデフォルト・リストにあるかを調べるには、sp_helpdevice を使用してください。

注意 システム管理者は、パフォーマンス統計およびその他の考慮事項に基づいて、一定の記憶領域の割り付けを行っている場合があります。データベースを作成する前に、システム管理者に確認してください。

このデフォルトのロケーションに格納するデータベースに 5MB のサイズを指定するには、次のように on default = *size* を使用します。

```
use master
create database newpubs
on default = 5
drop database newpubs
use pubs2
```

このデータベースに対して別のロケーションを指定するには、データベースを格納するデータベース・デバイスの論理名を指定します。デバイスごとに異なる量の領域を指定して、1 つのデータベースを複数のデータベース・デバイスに格納できます。

次の例では、newpubs データベースを作成して、pubsdata 上に 3MB、newdata 上に 2MB を割り付けます。

```
create database newpubs
on pubsdata = 3, newdata = 2
```

on 句とサイズを省略すると、データベースは、sysdevices に示されるデフォルト・データベース・デバイスのプールから 2MB のサイズで作成されます。

データベース割り付けのサイズは、2MB から 2^{23}MB の範囲です。

log on 句

作成するデータベースが非常に小さい、重要度の低いものでないかぎり、create database コマンドには必ず log on database_device 拡張機能を使用してください。これによって別のデータベース・デバイスにトランザクション・ログが置かれます。別のデバイスにログを置くには、次のような理由があります。

- dump database だけでなく dump transaction も使用できるため、時間とテープを節約できます。
- ログに固定サイズを確立でき、他のデータベース・アクティビティと領域を取り合わなくてすみます。
- パフォーマンスが向上します。
- ハード・ディスクが故障した場合にフル・リカバリが保証されます。

次のコマンドは、newpubs のログを論理デバイス pubslog に、1MB のサイズで置きます。

```
create database newpubs
on pubsdata = 3, newdata = 2
log on pubslog = 1
```

注意　log on 拡張機能を使用すると、“logsegment” というセグメントにデータベース・トランザクション・ログが置かれます。既存のログに領域を追加するには、alter database と、場合によっては sp_extendsegment を使用します。詳細については、『リファレンス・マニュアル：コマンド』、『リファレンス・マニュアル：プロシージャ』、または『システム管理ガイド 第 2 巻』の「第 8 章 セグメントの作成と使用」を参照してください。

トランザクション・ログに必要なデバイスのサイズは、更新アクティビティの量やトランザクション・ログ・ダンプの頻度によって異なります。一般的なガイドラインとしては、データベースに割り付けた領域の 10 パーセントから 25 パーセントをログに割り付けます。

for load オプション

オプションの for load 句は、データベース・ダンプのロードだけに使用できる create database の簡易版を呼び出します。for load オプションは、メディア障害からのリカバリや、あるマシンから別のマシンへのデータベースの移動に使用します。『リファレンス・マニュアル：コマンド』、『システム管理ガイド 第 2 巻』の「第 12 章 ユーザ・データベースのバックアップとリストア」を参照してください。

quiesce database コマンド

次のコマンドを使用して、データベースをスリープ・モードにすることができます。

```
quiesce database
```

このコマンドは、指定されたデータベースのリストへの更新処理を中断し再開します。『リファレンス・マニュアル：コマンド』、『システム管理ガイド：第 2 巻』の「第 11 章 バックアップおよびリカバリ・プランの作成」に説明されている「データベース更新のサスペンドと再開」を参照してください。

データベースのサイズの変更

データベースに割り付けられた記憶領域がいっぱいになると、新しいデータの追加やデータベースの更新ができなくなります。既存のデータは常に保護されます。データベースに割り付けられた領域が少なすぎることがわかった場合、データベース所有者は、alter database コマンドを使用して領域を増やすことができます。alter database パーミッションは、デフォルトではデータベース所有者に付与されますが、譲渡はできません。alter database を使用するには、master データベースを使用している必要があります。

デフォルトの増加幅は、領域のデフォルト・プールからの 2MB です。次の文はデフォルト・データベース・デバイスの newpubs に 2MB を追加します。

```
alter database newpubs
```

『リファレンス・マニュアル：コマンド』を参照してください。

alter database コマンド中の on 句は、create database の on 句によく似ています。for load 句は create database コマンド中の for load 句によく似ており、for load 句を使用して作成されたデータベースでのみ使用できます。

newpubs に割り付けられた領域を、データベース・デバイス pubsdata 上で 2MB、およびデータベース・デバイス newdata 上で 3MB 増やすには、次を入力します。

```
alter database newpubs
on pubsdata = 2, newdata = 3
```

alter database を使用して、データベースですでに使用されているデバイスに追加の領域を割り付けると、すでにそのデバイス上にあるセグメントはすべて、追加の領域フラグメントを使用します。既に既存のセグメントにマップされているオブジェクトは、すべて追加の領域に増大できるようになります。データベースの最大セグメント数は 32 です。

alter database を使用して、データベースでまだ使用されていないデバイスに領域を割り付けると、system と default セグメントが新しいデバイスにマップされます。このセグメント・マッピングを使用するには、sp_dropsegment を使用してデバイスから不要なセグメントを削除します。『リファレンス・マニュアル：プロシージャ』を参照してください。

注意　sp_extendsegment を使用すると、system セグメントと default セグメントのマップが自動的に解除されます。

---

with override の詳細については、『システム管理ガイド 第 2 巻』の「第 6 章　ユーザ・データベースの作成と管理」を参照してください。

データベースの削除

データベースを削除するには、drop database コマンドを使用します。drop database　コマンドは、指定されたデータベースとそのすべての内容を Adaptive Server から削除し、そのデータベースに割り付けられていた記憶領域を解放し、そのデータベースへの参照を master データベースから削除します。

『リファレンス・マニュアル：コマンド』を参照してください。

使用中のデータベース、つまりユーザが読み書きのために開いているデータベースは、削除できません。

1 つのコマンドで複数のデータベースを削除できます。次に例を示します。

```
drop database newpubs, newdb
```

損傷しているデータベースは drop database で削除できます。drop database が機能しない場合は、dbcc dbrepair を使用して、損傷しているデータベースを修復してから削除してください。

テーブルの作成

テーブルを作成するときに、カラムに名前を付けてそれぞれにデータ型を指定します。また、特定のカラムに null 値を保持するかどうかを指定したり、テーブルのカラムに整合性制約を指定したりできます。データベースあたり 20 億のテーブルを作成できます。

オブジェクト名または識別子の長さに対する制限は、通常の識別子の場合に 255 バイト、区切り識別子の場合に 253 バイトです。この制限は、テーブル名、カラム名、インデックス名などのほとんどのユーザ定義識別子に適用されます。

変数では “@” が 1 バイトとしてカウントされるため、変数名は最大で 254 文字です。

テーブルあたりの最大カラム数

1 つのテーブルの最大カラム数は、サーバの論理ページ・サイズや、テーブルが全ページ・ロックとデータオンリー・ロックのどちらで設定されるかなど、さまざまな要因によって決まります。『リファレンス・マニュアル：コマンド』を参照してください。

例

これらの例を試すには、前の項で作成した newpubs データベースを使用してください。そうしないと、ここで行う変更が、pubs2 や pubs3 などの他のデータベースに影響してしまいます。

create table の最も簡単な形式は、次のとおりです。

```
create table table_name
(column_name datatype)
```

たとえば、“some_name” という名前の固定長 11 バイトのカラムを 1 つ持つ names というテーブルを作成するには、次のように入力します。

```
create table names
(some_name char(11))
drop table names
```

set quoted_identifier on を設定した場合、テーブル名とカラム名には区切り識別子を使用できます。設定していない場合は、「識別子」(10 ページ) に説明されている識別子の規則に従ってください。カラム名は 1 つのテーブル内ではユニークである必要がありますが、同じデータベース内の別のテーブルでは、同じカラム名を使用できます。

カラムにはそれぞれデータ型が必要です。前述の例では、カラム名の後の “char” というワードが、カラムのデータ型、つまりそのカラムが持つ値の型を表します。データ型については、「第 6 章 データ型の使用と作成」を参照してください。

データ型の後のカッコで囲まれた数字は、カラムに格納できる最大バイト数を示します。データ型のなかには、最大長を指定するものがあります。システム定義の最大長を持つものもあります。

カラム名のリストはカッコで囲み、それぞれのカラム定義の後にはカンマを置きます。最後のカラム定義の後には、カンマを付ける必要はありません。

注意 create table 文に default が含まれている場合、その default 内で変数を使用することはできません。

---

create table コマンドの詳細については、『リファレンス・マニュアル：コマンド』を参照してください。

テーブル名の選択

create table コマンドは、現在開かれているデータベースに新しいテーブルを構築します。テーブル名はユーザごとにユニークである必要があります。

create table 文中で、テーブル名の前にシャープ記号 (#) を置くか、または “tempdb..” というプレフィクスを指定するかして、テンポラリ・テーブルを作成できます。詳細については、「テンポラリ・テーブルの使用」(274 ページ) を参照してください。

自分で作成したテーブルやその他のオブジェクトは、名前を修飾せずに使用できます。また、データベース所有者が作成したオブジェクトも、適切なパーミッションを持ってさえいれば、名前を修飾せずに使用できます。これらの規則は、システム管理者およびデータベース所有者を含むすべてのユーザに適用されます。

複数のユーザが同じ名前のテーブルを作成できます。たとえば、“jonah” というユーザと “sally” というユーザが、それぞれ info というテーブルを作成できます。この 2 つのテーブルの両方に対するパーミッションを持っているユーザは、それぞれを jonah.info、sally.info として修飾する必要があります。Sally は Jonah のテーブルへの参照を、jonah.info として修飾する必要がありますが、自分のテーブルは単に info として参照できます。

異なるデータベースでのテーブルの作成

テーブル名をデータベースの名前で修飾することによって、現在のデータベース以外のデータベースでテーブルを作成できます。ただし、テーブルを作成するデータベースで認可されたユーザである必要があり、そのデータベースでの create table パーミッションを持っている必要があります。

pubs2 または pubs3 を使用していて、newpubs という別のデータベースがある場合、次のようにして、newtab というテーブルを newpubs に作成できます。

```
create table newpubs..newtab (col1 int)
```

現在のデータベース以外のデータベースに、ビュー、ルール、デフォルト、ストアド・プロシージャ、トリガなど、他のデータベース・オブジェクトを作成することはできません。

create table 構文

create table 文は次のことを実行します。

- テーブル内の各カラムを定義します。
- カラムの名前とデータ型を提供し、各カラムでの null 値の扱いを指定します。
- カラムがある場合は、どのカラムに IDENTITY プロパティを含めるかを指定します。
- カラムレベルの整合性制約とテーブルレベルの整合性制約を定義します。各テーブル定義には、カラムごと、およびテーブルごとに複数の制約を含めることができます。

たとえば、pubs2 データベースにある titles テーブルの create table 文は、次のようになります。

```
create table titles
(title_id tid,
title varchar(80) not null,
type char(12),
pub_id char(4) null,
price money null,
advance money null,
royalty int null,
total_sales int null,
notes varchar(200) null,
pubdate datetime,
contract bit not null)
```

『リファレンス・マニュアル：コマンド』を参照してください。

以降の項では、システム提供のデータ型、ユーザ定義データ型、null 型、IDENTITY カラムなど、テーブル定義のコンポーネントについて説明します。

注意 create table の拡張構文である on *segment_name* を使用して、既存のセグメントにテーブルを配置できます。*segment_name* は、特定のデータベース・デバイスまたはデータベース・デバイスの集合を指します。セグメントにテーブルを作成する前に、使用できるセグメントのリストについて、システム管理者またはデータベース所有者に確認してください。パフォーマンス上の理由やその他の考慮事項により、特定のテーブルまたはインデックスに一定のセグメントが割り付けられている場合があります。

IDENTITY カラムの使用

IDENTITY カラムには、Adaptive Server によって自動的に生成される、テーブル内で各ローをユニークに識別する値が含まれています。

各テーブルは IDENTITY カラムを 1 つだけ持つことができます。IDENTITY カラムは、create table 文や select into 文でテーブルを作成するときに定義するか、alter table 文を使用して後から追加することができます。

IDENTITY カラムは、create table 文に、null または not null ではなく、キーワード identity を指定して定義します。IDENTITY カラムは、データ型が numeric で位取りが 0、または任意の整数型である必要があります。新しいテーブルでは、1 桁から 38 桁の任意の精度で IDENTITY カラムを定義します。

```
create table table_name
    (column_name numeric(precision ,0) identity)
```

カラムに指定できる最大の値は $10^{precision} - 1$ です。たとえば、次のコマンドは、IDENTITY カラムに最大 $10^5 - 1$ (= 9999) の値を定義できるテーブルを作成します。

```
create table sales_daily
    (sale_id numeric(5,0) identity,
    stor_id char(4) not null)
```

IDENTITY カラムが最大値に達すると、後続の insert 文はすべてエラーとなり、現在のトランザクションはアボートします。

auto identity データベース・オプションと size of auto identity 設定パラメータを使用して、自動 IDENTITY カラムを作成できます。ユニークでないインデックスに IDENTITY カラムを含めるには、identity in nonunique index データベース・オプションを使用します。

注意 デフォルトでは、Adaptive Server は、ローの番号付けを値 1 で開始し、ローが追加されるごとにそれに続けて番号付けをします。手作業による挿入、削除、またはトランザクション・ロールバックなどのいくつかのアクティビティ、およびサーバの停止や障害によって、IDENTITY カラム値にギャップが発生することがあります。Adaptive Server は、「テーブルの identity ギャップの管理」(277 ページ) で説明する identity ギャップを制御するメソッドをいくつか提供しています。

ユーザ定義データ型による IDENTITY カラムの作成

ユーザ定義データ型を使用して IDENTITY カラムを作成できます。ユーザ定義データ型は、numeric 型で位取りが 0 である基本の型を持っているか、任意の整数型である必要があります。IDENTITY プロパティを持つユーザ定義データ型が作成されている場合は、カラム作成時に identity キーワードを繰り返す必要がありません。

次の例は、IDENTITY プロパティを持つユーザ定義データ型を示しています。

```
sp_addtype ident, "numeric(5)", "identity"
```

次の例は、ident データ型に基づいた IDENTITY カラムを示しています。

```
create table sales_monthly
    (sale_id ident, stor_id char(4) not null)
```

ユーザ定義のデータ型が not null で作成されている場合は、create table 文で identity キーワードを指定する必要があります。null 値を許可するユーザ定義データ型からは IDENTITY カラムを作成できません。

IDENTITY カラムの参照

参照されるカラムを作成するときと同様、IDENTITY カラムを参照するテーブル・カラムを作成するときは、そのデータ型の定義が IDENTITY カラムのデータ型定義と同じであることを確認してください。たとえば pubs3 データベースでは、sales テーブルは ord_num カラムを IDENTITY カラムとして使用して定義されています。

```
create table sales
(stor_id char(4) not null
      references stores(stor_id),
ord_num numeric(6,0) identity,
date datetime not null,
unique nonclustered (ord_num))
```

ord_num IDENTITY カラムは一意性制約として定義されています。これは、このカラムが salesdetail の ord_num カラムを参照するのに必要です。salesdetail は次のように定義されています。

```
create table salesdetail
(stor_id char(4) not null
     references storesz(stor_id),
ord_num numeric(6,0)
     references salesz(ord_num),
title_id tid not null
     references titles(title_id),
qty smallint not null,
discount float not null)
```

sales にローを挿入してから salesdetail にローを挿入する簡単な方法は、*@@identity* グローバル変数を使用して salesdetail に IDENTITY カラムを挿入することです。@@identity グローバル変数は、生成された最新の IDENTITY カラム値を使用します。次に例を示します。

```
begin tran
insert sales values ("6380", "04/25/97")
insert salesdetail values ("6380", @@identity, "TC3218", 50, 50)
commit tran
```

この例は、2 つの挿入が成功するために相互依存するため、1 つのトランザクションになります。たとえば sales の挿入が失敗すると、*@@identity* の値は異なるものとなり、間違ったローが salesdetail に挿入されることになります。2 つの挿入は 1 つのトランザクション内にあるので、いずれかが失敗すると、トランザクション全体が拒否されます。

「@@identity を使用した IDENTITY カラム値の取得」(222 ページ) を参照してください。トランザクションの詳細については、「第 23 章 トランザクション：データの一貫性およびリカバリ」を参照してください。

syb_identity による IDENTITY カラムの参照

IDENTITY カラムを一度定義すれば、実際のカラム名を覚えておく必要はありません。テーブルの select、insert、update、または delete 文内で、syb_identity キーワードを必要に応じてテーブル名で修飾して使用できます。

たとえば次のクエリは、*sale_id* が 1 であるローを選択します。

```
select * from sales_monthly
    where syb_identity = 1
```

「隠し」IDENTITY カラムの自動作成

システム管理者は、auto identity データベース・オプションを使用すると、新しいテーブルに 10 桁の IDENTITY カラムを自動的に組み込むことができます。この機能を有効にするには、次のコマンドを使用します。

```
sp_dboption database_name, "auto identity", "true"
```

ユーザがプライマリ・キー、一意性制約、または IDENTITY カラムを指定しないで新しいテーブルを作成するたびに、Adaptive Server は自動的に IDENTITY カラムを定義します。IDENTITY カラムは、select * を使用してテーブルのすべてのカラムを取り出しても、参照できません。カラム名 SYB_IDENTITY_COL (すべて大文字) を、明示的に select リストに含める必要があります。コンポーネント統合サービスが有効になっている場合、プロキシ・テーブルの自動 IDENTITY カラムは OMNI_IDENTITY_COL と呼ばれます。

自動 IDENTITY カラムの精度を設定するには、size of auto identity 設定パラメータを使用します。たとえば、IDENTITY の精度を 15 に設定するには、次のコマンドを使用します。

```
sp_configure "size of auto identity", 15
```

カラムにおける null 値の許可

create table 文で null または not null を省略すると、Adaptive Server はデータベースに定義された null モード (デフォルトでは not null) を使用します。allow nulls by default オプションを true に設定する場合は、sp_dboption を使用します。

not null と定義されたカラムには入力を行う必要があります。行わない場合は、エラー・メッセージが表示されます。「「不定の値」：null」(61 ページ) を参照してください。

カラムを null として定義すると、不定のデータにはプレースホルダが提供されます。たとえば titles テーブルでは、price、advance、royalty、および total_sales が、null 入力可能なように設定されています。

しかし、title_id と title はそのように設定されていません。これらのカラムのエントリがないのは意味がありませんし、紛らわしいためです。タイトルなしの価格というのは意味を成しませんが、価格がないタイトルというのは、単に価格が未設定であるか、または現在のところ不明であるということを意味します。

カラム内の情報が他のカラムの意味に重要な影響を持つ場合は、create table の中で not null を使用してください。

null 値に使用される制約およびルール

null 値を入力できるようにカラムを定義してから、null 値を禁止する制約やルールでその定義を上書きすることはできません。たとえば、カラム定義が null を指定し、ルールが次の指定を行うとします。

```
@val in (1,2,3)
```

この場合、暗黙的または明示的な null はルールに違反しません。ルールで次のように指定されていても、カラム定義によってルールは無効になります。

```
@val is not null
```

制約の詳細については、「テーブルの整合性制約の定義」(283 ページ) を参照してください。ルールについては、「第 14 章 データのデフォルトとルールの定義」を参照してください。

デフォルトと null 値

null カラムと not null カラムの両方で、デフォルト、つまり入力が行われなかった場合に自動的に提供される値を使用できます。デフォルトは入力とみなされます。ただし、not null カラムに null デフォルトを指定することはできません。create table の default 制約を使用するか、create default を使用して、null 値をデフォルトに指定できます。default 制約については、この章の後半で説明します。create default については、「第 14 章 データのデフォルトとルールの定義」を参照してください。

カラムを作成するときに not null を指定してそのデフォルトを作成しなかった場合、挿入時にユーザがそのカラムへの入力を行わないと、エラー・メッセージが発生します。さらに、null の値を使用してこのようなカラムに insert や update を実行することはできません。

表 8-1 は、ユーザがカラム値を指定しなかった場合や明示的に null 値を指定した場合の、カラムのデフォルトとその null 型との関係を示します。結果としては、カラムへの null 値、カラムへのデフォルト値、エラー・メッセージの 3 通りが考えられます。

表 8-1: カラム定義と null デフォルト

カラム定義	ユーザ入力	結果
null とデフォルトが定義されている	値を入力しない	デフォルトを使用
	null 値を入力	null を使用
null が定義され、デフォルトは定義されていない	値を入力しない	null を使用
	null 値を入力	null を使用
not null で、デフォルトが定義されている	値を入力しない	デフォルトを使用
	null 値を入力	null を使用
not null が定義され、デフォルトが定義されていない	値を入力しない	エラー
	null 値を入力	エラー

可変長データ型を必要とする null

null 値を格納できるのは、可変長データ型のカラムだけです。固定長データ型で null カラムを作成すると、Adaptive Server はこのカラムを対応する可変長データ型に変換します。Adaptive Server は、型の変更をユーザに通知しません。

表 8-2 は、固定長データ型と、それらを Adaptive Server が変換する可変長データ型のリストです。moneyn などの特定の可変長データ型は予約されています。それらを使用してカラム、変数、またはパラメータを作成することはできません。

表 8-2: 固定長データ型から可変長データ型への変換

元の固定長データ型	変換後のデータ型
char	varchar
nchar	nvarchar
unichar	univarchar
binary	varbinary
datetime	datetimn
float	floatn
bigint、int、smallint、tinyint	intn
unsigned bigint、unsigned int、unsigned smallint	uintn
decimal	decimaln
numeric	numericn
money、smallmoney	moneyn

char、nchar、unichar、binary カラムに入力されたデータについては、カラムが指定の長さになるまでスペースや 0 を埋め込むのではなく、可変長カラムの規則に従います。

text、unitext、image カラム

insert と null で作成された text、unitext、image カラムは、初期化されず、値を含んでいません。これらのカラムは記憶領域を使用せず、readtext または writetext と関連付けることはできません。

text、unitext、または image カラムに update を使用して null 値が書き込まれると、カラムが初期化され、そのカラムへの有効なテキスト・ポインタがテーブルに挿入されて、カラムに 2K データ・ページが割り付けられます。カラムは、一度初期化されると、readtext および writetext でアクセスできるようになります。『リファレンス・マニュアル：コマンド』を参照してください。

テンポラリ・テーブルの使用

テンポラリ・テーブルは tempdb データベース内に作成されます。テンポラリ・テーブルを作成するには、tempdb での create table パーミッションが必要です。create table パーミッションは、デフォルトではデータベース所有者にあります。

テンポラリ・テーブルには次の 2 つのタイプがあります。

- Adaptive Server セッション間で共有できるテーブル

  共有テンポラリ・テーブルを作成するには、create table 文の中で、テーブル名の一部として tempdb を指定します。たとえば、次に示す文は、Adaptive Server セッション間で共有できるテンポラリ・テーブルを作成します。

  ```
  create table tempdb..authors
  (au_id char(11))
  drop table tempdb..authors
  ```

  Adaptive Server は、このようにして作成されたテンポラリ・テーブルの名前を変更しません。テーブルは、現在のセッションが終了するまで、または所有者が drop table を使用して削除するまで存在します。

- 現在の Adaptive Server セッションまたはプロシージャによってのみアクセス可能なテーブル

  非共有テンポラリ・テーブルを作成するには、create table 文の中で、テーブル名の前にシャープ記号 (#) を指定します。次に例を示します。

  ```
  create table #authors
  (au_id char(11))
  ```

  テーブルは、現在のセッションやプロシージャが終了するまで、または所有者が drop table を使用して削除するまで存在します。

テーブル名の前にシャープ記号や “tempdb..” を使用せず、現在 tempdb を使用していない場合、テーブルは永久テーブルとして作成されます。永久テーブルは、所有者によって明示的に削除されるまで、データベース内に存在します。

次の文は、非共有テンポラリ・テーブルを作成します。

```
create table #myjobs
(task char(30),
start datetime,
stop datetime,
notes varchar(200))
```

このテーブルは、今日 1 日の仕事や用事のリストとして、開始と終了の記録やコメントなども書き込んで使用できます。このテーブルとそのデータは、現在の作業セッションが終了すると自動的に削除されます。テンポラリ・テーブルはリカバリできません。

ルール、デフォルト、およびインデックスをテンポラリ・テーブルに関連付けることはできますが、テンポラリ・テーブルにビューを作成したり、トリガを関連付けたりすることはできません。テンポラリ・テーブルを作成するときにユーザ定義データ型を使用できるのは、そのデータ型が tempdb..systypes に存在する場合にかぎります。

現在のセッションだけで tempdb にオブジェクトを追加するには、tempdb を使用している間に sp_addtype を実行します。オブジェクトを永続的に追加するには、model 内で sp_addtype を実行してから、Adaptive Server を再起動して、model が tempdb にコピーされるようにします。

テンポラリ・テーブル名がユニークであることの保証

テンポラリ・テーブルの名前が現在のセッションでユニークになるようにするために、Adaptive Server は次のことを行います。

- 必要に応じて、テーブル名を 238 バイトにトランケートする (シャープ記号 (#) を含む)。
- Adaptive Server セッションでユニークである 17 桁の数値サフィックスを追加する。

次の例は、#temptable として作成され、#temptable00000050010721973 として格納されるテーブルを示します。

```
use pubs2
go
create table #temptable (task char(30))
go
use tempdb
go
select name from sysobjects where name like
    "#temptable%"
go

name
------------------------------
#temptable00000050010721973
```

```
(1 row affected)
```

ストアド・プロシージャ内でのテンポラリ・テーブルの操作

ストアド・プロシージャは、現在のセッション中に作成されたテンポラリ・テーブルを参照できます。

“#” で始まる名前を持つテンポラリ・テーブル

ストアド・プロシージャ内で作成された、“#” で始まる名前を持つテンポラリ・テーブルは、プロシージャが終了すると削除されます。1 つのプロシージャで次のことができます。

- テンポラリ・テーブルの作成
- テーブルへのデータの挿入
- テーブルでのクエリの実行
- テーブルを参照する他のプロシージャの呼び出し

テンポラリ・テーブルを参照するプロシージャを作成するためには、テンポラリ・テーブルが存在する必要があるので、次の手順に従ってください。

1 create table を使用してテンポラリ・テーブルを作成します。

2 テンポラリ・テーブルにアクセスするプロシージャを作成しますが、テーブルを作成するプロシージャは作成しません。

3 テンポラリ・テーブルを削除します。

4 テーブルの作成、および手順 2 で作成したプロシージャの呼び出しを実行するプロシージャを作成します。

tempdb で始まる名前を持つテンポラリ・テーブル

ストアド・プロシージャ内から create table tempdb..tablename を使用して、# プレフィクスを使用しないテンポラリ・テーブルを作成できます。このようなテーブルはプロシージャが完了しても削除されないので、独立したプロシージャで参照できます。このようなテーブルを作成するには、前述の手順に従います。

警告！ ユーザ間およびセッション間でテーブルを共有する場合にのみ、ストアド・プロシージャ内から “tempdb..” プレフィクス付きのテンポラリ・テーブルを作成します。テンポラリ・テーブルを作成して削除するストアド・プロシージャは、# プレフィクスを使用して、テーブルが不用意に共有されないようにします。

テンポラリ・テーブルについての一般的な規則

で始まる名前を持つテンポラリ・テーブルには、次の制限があります。

- これらのテーブルにはビューを作成できません。
- これらのテーブルにトリガを関連付けることはできません。
- ストアド・プロシージャ内から、以下は実行できません。
  a　テンポラリ・テーブルの作成
  b　テンポラリ・テーブルの削除
  c　同じ名前の新しいテンポラリ・テーブルの作成
- どのセッションまたはプロシージャがこれらのテーブルを作成したかは区別できません。

このような制限は、tempdb 内で作成された、共有可能な、テンポラリ・テーブルには適用されません。

いずれのタイプのテンポラリ・テーブルにも適用される規則は、次のとおりです。

- ルール、デフォルト、およびインデックスをテンポラリ・テーブルに関連付けることができます。テンポラリ・テーブルが削除されると、テンポラリ・テーブル上に作成されたインデックスも削除されます。
- sp_help などのシステム・プロシージャは、tempdb から呼び出した場合にだけ、テンポラリ・テーブルに対して機能します。
- ユーザ定義データ型は、そのデータ型が tempdb にないかぎり、テンポラリ・テーブルでは使用できません。つまり、Adaptive Server が最後に再起動されてから、そのデータ型が tempdb 内で明示的に作成されていないかぎり、ユーザ定義データ型をテンポラリ・テーブル内で使用することはできません。
- テンポラリ・テーブルに select into を行うために select into/bulkcopy オプションを on に設定する必要はありません。

テーブルの identity ギャップの管理

IDENTITY カラムには、テーブル内のローごとに、Adaptive Server が生成するユニークな ID 番号が含まれています。サーバがデフォルトで ID 番号を生成する方法によって、ID 番号に大きなギャップが発生することがあります。identity_gap パラメータを使用すると、特定のテーブルで ID 番号を制御でき、発生する可能性のあるギャップを制御できます。

デフォルトでは、Adaptive Server は、それぞれの ID 番号を必要に応じてディスクに書き込むのではなく、ID 番号のブロックをメモリに割り付けます。各番号を書き込む方法は、処理に時間がかかります。サーバは、各ブロックの一番大きな番号を、テーブルのオブジェクト・アロケーション・マップ (OMA) ページに書き込みます。この番号は、現在割り付けられている番号のブロックが使用されたり、「消去」されたりした後の、次のブロックの開始ポイントとして使用されます。ブロックのその他の番号はメモリ内に保持されますが、ディスクには保存されません。番号は、メモリに割り付けられた時点で消去されたとみなされます。そして、ローに割り当てられたか、システム障害などなんらかの異常事態によってメモリから消去されたかの理由によって、メモリから削除されます。

ID 番号のブロックを割り付けると、テーブルの競合が減るため、パフォーマンスが向上します。しかし、すべての ID 番号が割り当てられる前にサーバが障害を起こしたり、no wait で停止したりすると、使用されていない番号が消去されてしまいます。サーバは、再度稼働するときに、ディスクに書き込んだ直前のブロックの最も大きい数字に基づいて、次のブロックの番号付けを開始します。障害の前にいくつの割り付け番号がローに割り当てられたかによって、ID 番号に大きなギャップが発生することがあります。

また、identity ギャップは、アクティブなデータベースをダンプしたりロードしたりした結果として発生する場合もあります。データベースをダンプすると、データベース・オブジェクトはオブジェクト・アロケーション・マップ・ページに保存されます。オブジェクトが現在使用されている場合、maximum used identity value は OAM ページにないため、ダンプされません。

identity ギャップを制御するパラメータ

Adaptive Server は、表 8-3 に示すように、ID 番号のギャップを制御できるパラメータを提供しています。

表 8-3: identity ギャップを制御するパラメータ

パラメータ名	範囲	何と使用するか	説明
identity_gap	テーブル固有	create table または select into	特定のテーブルの特定のサイズの ID 番号ブロックを作成する。テーブルの identity burning set factor を上書きする。identity grab size とともに機能する。

パラメータ名	範囲	何と使用するか	説明
identity burning set factor	サーバワイド	sp_configure	各ブロックに割り付ける、使用可能な ID 番号の合計のパーセンテージを示す。identity grab size とともに機能する。テーブルの identity_gap が 1 以上に設定されている場合、identity burning set factor はそのテーブルには作用しない。burning set factor は、identity_gap が 0 に設定されているすべてのテーブルに使用される。 identity burning set factor を設定するときは、数字を小数で表してから 10,000,000 (10^7) を掛けて、sp_configure で使用する正しい値を取得する。たとえば潜在的な IDENTITY カラムの値の 15 パーセント (.15) を一度に解放するには、.15 x 10^7 (つまり 1,500,000) を指定する。 sp_configure “identity burning set factor”, 1500000
identity grab size	サーバワイド	sp_configure	プロセスごとに連続する ID 番号のブロックを予約する。identity burning set factor および identity_gap とともに機能する。

identity burning set factor と identity_gap の比較

identity_gap パラメータを使用すると、特定のテーブルの identity ギャップのサイズを制御できます。

たとえば、書店のすべての本を含む books という名前のテーブルを作成する場合、各本にはユニークな ID 番号が必要で、これは Adaptive Server が自動的に生成します。books には IDENTITY カラムが含まれ、デフォルトの数値 (18, 0) を使用し合計で 999,999,999,999,999,999 個の ID 番号を付けることができます。identity burning set factor 設定パラメータには、デフォルト設定の 5000 (999,999,999,999,999,999 の .05 パーセント) を使用します。これは、Adaptive Server が 500,000,000,000,000 個の番号のブロックを割り付けることを意味します。

サーバは最初の 500,000,000,000,000 個の番号をメモリに割り付け、ブロックの最も大きな番号 (500,000,000,000,000) をテーブルのオブジェクト・アロケーション・マップ・ページに格納します。すべての番号がローに割り当てられるかまたは消去されると、Adaptive Server は 500,000,000,000,001 で始まる次のブロック (次の 500,000,000,000,000) を取得し、ブロックの最も大きな番号として 1,000,000,000,000,000 を格納します。

サーバがロー番号 500,000,000,000,022 の後で失敗すると、books の ID 番号として、1 ～ 500,000,000,000,022 の番号だけが使用されます。500,000,000,000,023 から 1,000,000,000,000,000 の番号は消去されています。Adaptive Server が再度稼働すると、テーブルのオブジェクト・アロケーション・マップ・ページに格納されている最も大きな番号に 1 を加えたもの (1,000,000,000,000,001) から ID 番号の作成を開始します。これによって 499,999,999,999,978 個の ID 番号のギャップが発生します。

ID 番号のギャップの削減

identity_gap 値が 1000 である books テーブルを (上述の例から) 作成します。これは、500,000,000,000,000 個の ID 番号のブロックとなったサーバワイドな identity burning set factor 設定を上書きします。代わりに、ID 番号は 1000 のブロックでメモリに割り付けられます。

サーバは最初の 1000 の番号を割り付け、ブロックの最も大きな番号 (1000) をディスクに格納します。すべての番号が使用されると、Adaptive Server は 1001 で始まる次の 1000 番号を取得し、最も大きな番号として 2000 を格納します。

Adaptive Server がロー番号 1002 の後で失敗すると、番号 1000 ～ 1002 を使用し、番号 1003 ～ 2000 は失われます。Adaptive Server を再起動すると、テーブルのオブジェクト・アロケーション・マップ・ページに格納されている最も大きな番号に 1 を加えたもの (2000) から ID 番号の作成を開始します。これによって発生するギャップは 998 個の番号だけです。

サーバワイドな table burning set factor を使用する代わりにテーブルに identity_gap を設定することによって、ID 番号のギャップを大幅に減らすことができます。しかし、この値を小さすぎる値に設定すると、サーバがブロックの最も大きな番号をディスクに書き込むたびに、パフォーマンスに影響します。たとえば、identity_gap を 1 に設定したとします。これは ID 番号を一度に 1 つ割り付けることを意味します。この場合、サーバはローが作成されるたびに新しい番号を書き込まなければならず、テーブルでページ・ロックの競合が発生するため、パフォーマンスが低下します。状況に合わせて、最も小さなギャップ値で最高のパフォーマンスを得るための最適な設定を見つけてください。

テーブル固有の identity ギャップの設定

create table または select into を使用してテーブルを作成するときに、テーブル固有の identity ギャップを設定します。

この文は、identity カラムを 1 つ持つ mytable というテーブルを作成します。

```
create table mytable (IdNum numeric(12,0) identity)
with identity_gap = 10
```

identity ギャップは 10 に設定されています。これは、メモリ内に 10 ブロック単位で ID 番号が割り付けられていることを示します。サーバが障害を起こしたり with no wait で停止したりした場合、ローに割り当てられた最後の ID 番号と、次にローに割り当てられる ID 番号の最大ギャップは、10 番号分です。

固有の identity ギャップが設定されているテーブルから、select into 文でテーブルを作成している場合、新しいテーブルは親テーブルから identity ギャップの設定を継承しません。代わりに、新しいテーブルは identity burning set factor 設定を使用します。新しいテーブルに固有の identity_gap 設定を指定するには、select into 文で identity ギャップを指定します。新しいテーブルには、親テーブルと同じ identity ギャップでも、異なる identity ギャップでも指定できます。

たとえば、identity ギャップを指定して、既存のテーブル (mytable) から新しいテーブル (newtable) を作成する場合は、次のように入力します。

```
select IdNum into newtable
with identity_gap = 20
from mytable
```

テーブル固有の identity ギャップの変更

特定のテーブルの identity ギャップを変更するには、次のように sp_chgattribute を使用します。

sp_chgattribute "*table_name*", "identity_gap", *set_number*

次に例を示します。

```
sp_chgattribute "mytable", "identity_gap", 20
```

identity_gap 設定ではなく identity burning set factor 設定を使用するように mytable を変更するには、次のように、identity_gap を 0 に設定します。

```
sp_chgattribute "mytable", "identity_gap", 0
```

『リファレンス・マニュアル：プロシージャ』を参照してください。

テーブル固有の identity ギャップ情報の表示

テーブルの identity_gap 設定を参照するには、sp_help を使用します。

たとえば、identity_gap カラムの値 0 (出力の終わりの方) は、テーブル固有の identity ギャップが設定されていないことを示します。mytable は、サーバワイドの identity burning set factor 値を使用します。

```
sp_help mytable
Name    Owner    Object_type    Create_date
------- -------- -------------- ------------------------------
mytable      dbo     user table            Nov 29 2004 1:30PM

(1 row affected)
. . .
exp_row_size reservepagegap fillfactor max_rows_per_page identity_gap
------------ -------------- ---------- ----------------- ------------
           1              0          0                 0            0
```

mytable の identity_gap を 20 に変更すると、テーブルの sp_help 出力は、identity_gap カラムに 20 と表示されます。この設定は、サーバワイドな identity burning set factor 値を上書きします。

```
sp_help mytable
Name    Owner    Object_type    Create_date
------- -------- -------------- ------------------------------
```

```
mytable dbo      user table     Nov 29 2004 1:30PM

(1 row affected)
. . .
exp_row_size reservepagegap fillfactor max_rows_per_page identity_gap
------------ -------------- ---------- ----------------- ------------
           1              0          0                 0           20
```

その他の原因によるギャップ

IDENTITY カラムへの手動による挿入、ローの削除、identity grab size 値の設定、およびトランザクション・ロールバックによって、IDENTITY カラムの値にギャップが発生することがあります。identity burning set factor の設定はこれらのギャップに影響しません。

たとえば、次の値を持つ IDENTITY カラムがあるとします。

```
select syb_identity from stores_cal

 id_col
 -------
       1
       2
       3
       4
       5
```

IDENTITY カラムが 2 と 4 の間であるすべてのローを削除できます。カラム値にはギャップが発生します。

```
delete stores_cal
where syb_identity between 2 and 4
select syb_identity from stores_cal

id_col
------
       1
       5
```

テーブルに対して identity_insert on を設定すると、テーブル所有者、データベース所有者、またはシステム管理者は、5 より大きい任意の有効な値を手動で挿入できます。たとえば、次のように、55 という値を挿入すると、IDENTITY カラム値に大きなギャップが発生します。

```
insert stores_cal
(syb_identity, stor_id, stor_name)
values (55, "5025", "Good Reads")
select syb_identity from stores_cal

id_col
 -------
       1
```

```
 5
55
```

その後、identity_insert が off に設定されると、Adaptive Server は、次の挿入で IDENTITY カラムに 55 + 1、つまり、値 56 を割り当てます。insert 文を含むトランザクションがロールバックされると、Adaptive Server は値 56 を廃棄して、次の挿入に 57 を使用します。

テーブル挿入が IDENTITY カラムの最大値に達した場合

テーブルに挿入できるローの最大数は、IDENTITY カラムの精度の設定によって異なります。テーブルが限界値に達した場合は、現在より大きな精度を指定してテーブルを再作成できます。または、テーブルの IDENTITY カラムが参照整合性に使用されていなければ、bcp を使用してギャップを取り除くことができます。「IDENTITY カラムの最大値を超えた場合」(224 ページ) を参照してください。

テーブルの整合性制約の定義

Transact-SQL では、データベース内のデータの整合性を維持するために、次の 2 つの方法を用意しています。

- ルール、デフォルト、インデックス、およびトリガを定義する
- create table 整合性制約を定義する

どちらの方法を選ぶかは、要件によって異なります。整合性制約によって、(SQL 規格で定義されているように) テーブル作成プロセスの間に 1 つの手順で整合性制御を定義するという利点、およびこのような整合性制御を作成するプロセスを簡易化するという利点が提供されます。しかし、整合性制約はスコープがさらに限定されており、デフォルト、ルール、インデックス、およびトリガほど包括的ではありません。

たとえばトリガが提供する参照整合性の処理は、create table で宣言されているものより複雑です。create table で定義される整合性制約はそのテーブルに固有なので、他のテーブルにはバインドできません。削除や変更を行うには、alter table を使用する必要があります。制約には、同じテーブル上でも、サブクエリや集合関数を入れることはできません。

2 つの方法は相互に排他的ではありません。整合性制約は、デフォルト、ルール、インデックス、およびトリガとともに使用できます。これによって、使用するアプリケーションの最適の方法の選択に柔軟性が与えられます。この項では、create table 整合性制約について説明します。デフォルト、ルール、インデックス、およびトリガは、後の章で説明します。

次のタイプの制約を作成できます。

- unique および primary key 制約では、テーブル内の指定のカラムで同じ値を持つローが 2 つ存在することはできません。さらに primary key 制約は、カラムのどのローにも null 値がないようにする必要があります。
- 参照整合性 (references) 制約は、特定のカラムに挿入されるデータは、指定のテーブルとカラムに一致するデータを既に持っている必要があります。テーブルの参照先テーブルを検出するには、sp_helpconstraint を使用します。
- check 制約 (検査制約) は、カラムに挿入されるデータの値を制限します。

カラム内での null 値の使用 (null または not null キーワード) を制限したり、カラムにデフォルト値 (default 句) を入れることによって、データ整合性を維持することもできます。null および not null キーワードの詳細については、「カラムにおける null 値の許可」(271 ページ)を参照してください。

テーブルに定義される制約の詳細については、「sp_helpconstraint によるテーブルの制約情報の表示」(332 ページ)を参照してください。

警告！ システム・テーブルに対して制約を定義したり、定義を変更したりしないでください。

テーブルレベルまたはカラムレベルの制約の指定

テーブルまたはカラムレベルで、整合性制約を宣言できます。その違いはユーザにはほとんどわかりませんが、カラムレベルの制約がチェックされるのは、カラム内の値が変更される場合のみであるのに対して、テーブルレベルの制約は、ローに対して何らかの変更が行われる場合にチェックされます。該当するカラムが変更されるかどうかは関係ありません。

カラムレベルの制約はカラム名とデータ型の後で、区切りカンマの前に置きます。テーブルレベルの制約は、カンマで区切られた個別の句として入力します。Adaptive Server は、テーブルレベルの制約とカラムレベルの制約を同じ方法で扱います。どちらも同様の効果があります。

ただし、複数のカラムに作用する制約は、テーブルレベルの制約として宣言する必要があります。たとえば、次の create table 文には、pub_id と pub_name の 2 つのカラムに作用する check 制約があります。

```
create table my_publishers
(pub_id      char(4),
pub_name     varchar(40),
constraint my_chk_constraint
    check (pub_id in ("1389", "0736", "0877")
        or pub_name not like "Bad News Books"))
```

単一のカラムで操作する制約はカラムレベルの制約として宣言できます。たとえば、次のように、前述の check 制約が 1 つのカラム (pub_id)) だけを使用する場合は、そのカラムに制約を置くことができます。

```
create table my_publishers
(pub_id     char(4) constraint my_chk_constraint
        check (pub_id in ("1389", "0736", "0877")),
pub_name     varchar(40))
```

カラムレベル制約、テーブルレベル制約いずれの場合も、constraint キーワードとそれに付随する *constraint_name* はオプションです。check 制約については、「検査制約の指定」(291 ページ) を参照してください。

注意　create table を検査制約付きで発行した後で、その同じバッチまたはプロシージャ内でテーブルにデータを挿入することはできません。create 文と insert 文を 2 つの異なるバッチまたはプロシージャに分けるか、execute を使用してアクションを別々に実行してください。

制約のエラー・メッセージの作成

sp_addmessage でエラー・メッセージを作成し、sp_bindmsg でメッセージを制約にバインドできます。

次に例を示します。

```
sp_addmessage 25001,
     "The publisher ID must be 1389, 0736, or 0877"
sp_bindmsg my_chk_constraint, 25001
insert my_publishers values
     ("0000", "Reject This Publisher")

Msg 25001, Level 16, State 1:
Server 'snipe', Line 1:
The publisher ID must be 1389, 0736, or 0877
Command has been aborted.
```

制約のメッセージを変更するには、新しいメッセージをバインドします。古いメッセージが新しいメッセージに置き換えられます。

メッセージを制約からバインド解除するには、sp_unbindmsg を使用します。ユーザ定義メッセージを削除するには、sp_dropmessage を使用します。

次に例を示します。

```
sp_unbindmsg my_chk_constraint
sp_dropmessage 25001
```

メッセージのテキストを変更するが同じエラー番号のままにしておくには、バインド解除してから sp_dropmessage で削除し、sp_addmessage でもう一度追加して、sp_bindmsg でバインドします。

検査制約の作成後

検査制約を作成すると、syscomments システム・テーブルの text カラムに、検査制約を記述する「ソース・テキスト」が格納されます。

警告！ syscomments からこの情報を削除しないでください。削除すると、Adaptive Server の今後のアップグレードで問題が発生する場合があります。

セキュリティ上の懸念がある場合は、『リファレンス・マニュアル：プロシージャ』で説明されているように、syscomments 内のテキストを、sp_hidetext を使用して暗号化します。「コンパイル済みオブジェクト」(3 ページ) を参照してください。

デフォルト・カラム値の指定

カラムレベルの整合性制約を定義する前に、カラムのデフォルト値を指定できます。「**default** 句」は、create table 文の一部として、カラムにデフォルト値を割り当てます。ユーザがカラムの値を入力しない場合、Adaptive Server はデフォルト値を挿入します。

default 句では、次の値を使用できます。

- *constant_expression* − カラムのデフォルト値として使用する定数式を指定します。定数式には、カラムや他のデータベース・オブジェクトの名前を使用することはできませんが、データベース・オブジェクトを参照しない組み込み関数は使用できます。このデフォルト値はカラムのデータ型と互換性がある必要があります。
- user − Adaptive Server がユーザ名をデフォルトとして挿入するように指定します。このデフォルトを使用するには、カラムのデータ型は char(30) または varchar(30) のいずれかである必要があります。
- null − Adaptive Server が null 値をデフォルトとして挿入するように指定します。not null キーワードを使用して、null 値を入力できないカラムにこのデフォルトを定義することはできません。

たとえば、次の create table 文は 2 つのカラム・デフォルトを定義します。

```
create table my_titles
(title_id        char(6),
title            varchar(80),
price            money       default null,
total_sales      int         default 0)
```

1 つのテーブル内のカラムごとに default 句を 1 つだけ含めることができます。

default 句を使用したデフォルトの割り当ては、Transact-SQL の 2 段階での方法よりも簡単です。Transact-SQL では、create default を使用してデフォルト値を宣言し、sp_bindefault を使用してカラムにバインドします。

一意性制約およびプライマリ・キー制約の指定

unique または primary key 制約を宣言して、指定のカラムに同じ値を持つローがテーブル内に 2 つとないことを保証できます。いずれの制約もユニーク・インデックスを作成してこのデータ整合性を維持します。ただし、primary key 制約の方が、unique 制約よりも制限的です。primary key 制約が指定されているカラムには、null 値を入れることができません。通常は、テーブルの primary key 制約を、他のテーブルで定義された参照整合性制約と組み合わせて使用します。

SQL 規格の unique 制約の定義は、カラム定義が null 値を使用しないことを指定します。デフォルトでは、カラム定義で null または not null キーワードを省略したときに、Adaptive Server は、null 値の入力を許可しないようにカラムを定義します (sp_dboption を使用してこれを変更していない場合)。Transact-SQL では、制約を実行するために使用するユニーク・インデックスによって、null 値の入力が許可されているため、unique 制約とともに、null 値の入力を許可するようカラムを定義できます。

注意　一意性制約とプライマリ・キー整合性制約を、sp_primarykey、sp_foreignkey、および sp_commonkey で定義されている情報と混同しないようにしてください。一意性制約と主キー制約は、実際にインデックスを作成して、テーブル・カラムのユニーク属性やプライマリ・キー属性を定義します。sp_primarykey、sp_foreignkey、および sp_commonkey は、(syskeys テーブルにある) テーブル・カラムのキーの論理関係を定義します。これは、ユーザがインデックスおよびトリガを作成することによって実行します。

---

unique 制約は、ユニーク・ノンクラスタード・インデックスをデフォルトで作成します。primary key 制約は、ユニーク・クラスタード・インデックスをデフォルトで作成します。どちらの制約でも、クラスタード・インデックスとノンクラスタード・インデックスのどちらでも宣言できます。

たとえば、次に示す create table 文は、テーブルレベルの unique 制約を使用して、stor_id カラムと ord_num カラムの 2 つのローが同じ値を持たないようにします。

```
create table my_sales
(stor_id     char(4),
ord_num      varchar(20),
date         datetime,
unique clustered (stor_id, ord_num))
```

1 つのテーブルに存在が可能なクラスタード・インデックスは 1 つだけであるため、指定できるのは unique clustered 制約または primary key clustered 制約のいずれか 1 つだけです。

unique 制約および primary key 制約を使用して、データ整合性を実行するときにユニーク・インデックス (with fillfactor、with max_rows_per_page、および on *segment_name* オプションを含む) を作成できます。ただし、インデックスには補足の機能があります。「第 13 章 テーブルのインデックスの作成」を参照してください。

参照整合性制約の指定

「参照整合性」とは、テーブル間の関係の管理に使用される手段です。テーブルを作成するときに、特定のカラムに挿入されるデータが、対応する値を別のテーブルに持つことを保証するための制約を定義できます。

テーブルに定義できる参照には、別のテーブルの参照、別のテーブルからの参照、自己参照 (同じテーブル内の参照) の 3 種類があります。

次に示す、pubs3 データベースの 2 つのテーブルは、宣言参照整合性がどのように機能するかを示します。1 つ目のテーブルの stores は、「参照先」テーブルです。

```
create table stores
(stor_id     char(4) not null,
stor_name    varchar(40) null,
stor_address varchar(40) null,
city         varchar(20) null,
state        char(2) null,
country      varchar(12) null,
postalcode   char(10) null,
payterms     varchar(12) null,
unique nonclustered (stor_id))
```

2 つ目のテーブルの store_employees は、stores テーブルへの参照を含んでいるので、「参照元」テーブルです。これは自己参照も含んでいます。

```
create table store_employees
(stor_id     char(4) null
       references stores(stor_id),
emp_id       id not null,
mgr_id       id null
       references store_employees(emp_id),
emp_lname    varchar(40) not null,
emp_fname    varchar(20) not null,
phone        char(12) null,
address      varchar(40) null,
city         varchar(20) null,
state        char(2) null,
country      varchar(12) null,
postalcode   varchar(10) null,
unique nonclustered (emp_id))
```

store_employees テーブルで定義されている参照は、次の制限を課します。

- store_employees テーブル内で指定された格納は、いずれも stores テーブルに含まれる必要があります。references 制約は、store_employees の stor_id カラムに挿入された値が my_stores の stor_id カラムに既に存在する必要があると示すことによって、これを実行します。
- すべての管理者は従業員 ID 番号を持つ必要があります。references 制約は、mgr_id カラムに挿入された値が emp_id カラムにすでに存在する必要があると示すことによって、これを実行します。

テーブルレベルまたはカラムレベルの参照整合性制約

テーブルまたはカラムレベルで、参照整合性制約を定義できます。前述の例の参照整合性制約は、create table 文中に references キーワードを使用して、カラムレベルで定義されていました。

テーブルレベルの参照整合性制約を定義するときは、foreign key 句と、複数のカラム名のリストを含めます。foreign key は、現在のテーブルにリストされたカラムが、後に続く references 句にリストされているカラムをターゲット・キーとする外部キーであることを指定します。次に例を示します。

```
constraint sales_detail_constr
     foreign key (stor_id, ord_num)
     references my_salesdetail(stor_id, ord_num)
```

foreign key 構文は、テーブルレベルの制約だけに使用できるもので、カラムレベルの制約には使用できません。「テーブルレベルまたはカラムレベルの制約の指定」(284 ページ) を参照してください。

カラムレベルかテーブルレベルで参照整合性制約を定義すると、sp_primarykey、sp_foreignkey、および sp_commonkey を使用して syskeys システム・テーブルにキーを定義できます。

テーブルが使用できる参照の最大数

1 つのテーブルに使用できる参照の最大数は 192 です。「sp_helpconstraint によるテーブルの制約情報の表示」(332 ページ) を参照してください。

create schema の使用による相互参照制約

まだ存在しないテーブルを参照するテーブルは作成できません。相互に参照する 2 つ以上のテーブルを作成するには、create schema を使用します。

「スキーマ」は、特定のユーザが所有するオブジェクトと、そのオブジェクトに関連付けられたパーミッションの集合です。create schema 文内のいずれかの文が失敗すると、コマンド全体が 1 つの単位としてロールバックされ、コマンドによる影響はありません。

create schema の構文は次のとおりです。

```
create schema authorization authorization name
create_object_statement
```

```
    [create_object_statement ...]
[permission_statement ...]
```

次に例を示します。

```
create schema authorization dbo
    create table list1
        (col_a char(10) primary key,
        col_b char(10) null
        references list2(col_A))
    create table list2
        (col_A char(10) primary key,
        col_B char(10) null
        references list1(col_a))
```

参照整合性制約を作成するための一般的な規則

テーブルに参照整合性制約を定義するときは、以下に従います。

- 参照先テーブルに対する references パーミッションを持っていることを確認します。『システム管理ガイド 第 1 巻』の「第 17 章 ユーザ・パーミッションの管理」を参照してください。
- 参照先カラムが、参照先テーブル内でユニーク・インデックスによる制約を持っていることを確認します。ユニーク・インデックスは、unique 制約か primary key 制約、または create index 文を使用して作成できます。たとえば、stores テーブルの参照先カラムは次のように定義されています。

  ```
  stor_id char(4) primary key
  ```

- 参照定義で使用されているカラムが、一致するデータ型を持っていることを確認します。たとえば、my_stores と store_employees の stor_id カラムは、いずれも char(4) データ型を使用して作成されています。store_employees の mgr_id カラムと emp_id カラムは、id データ型で作成されています。
- 参照先テーブルのカラムが、primary key 制約を介してプライマリ・キーとして指定されていれば、references 句内でカラム名を省略できます。
- 参照元テーブルに一致する値がある参照先テーブルに対して、ローの削除やカラム値の更新はできません。まず参照元テーブルから削除や更新を行ってから、参照先テーブルから削除、更新してください。

  同様に、参照先テーブルでは truncate table を使用できません。まず参照元テーブルをトランケートしてから、参照先テーブルをトランケートしてください。
- 参照元テーブルを削除から、参照先テーブルを削除してください。そうしないと、制約違反が発生します。
- テーブルの参照先テーブルを検出するには、sp_helpconstraint を使用します。

参照整合性制約を使用すると、トリガより簡単な方法でデータ整合性を維持できます。ただしトリガには、テーブル間の参照整合性を維持するための他の機能があります。「第 20 章 トリガ：参照整合性」を参照してください。

検査制約の指定

check 制約を宣言すると、ユーザがテーブル内のカラムに挿入する値を制限できます。検査制約は、限定された、特定の範囲の値を検査するアプリケーションに便利です。check 制約は、いずれの値もテーブルに挿入される前に渡されるよう、*search_condition* を指定します。*search_condition* には、次のものが含まれます。

- in によって導入される定数式のリスト
- between で指定される定数式の範囲。
- ワイルドカード文字を含む、like で提供される条件の集合

式は、算術演算と Transact-SQL 組み込み関数を含むことができます。*search_condition* には、サブクエリ、set 関数指定、またはターゲット指定を含めることはできません。

たとえば、次の文は、ある一定の値だけを pub_id カラムに入力できるようにします。

```
create table my_new_publishers
(pub_id      char(4)
        check (pub_id in ("1389", "0736", "0877",
             "1622", "1756")
        or pub_id like "99[0-9][0-9]"),
pub_name     varchar(40),
city         varchar(20),
state        char(2))
```

カラムレベルの検査制約は、制約が定義されているカラムだけを参照できます。テーブル内の他のカラムは参照できません。テーブルレベルの検査制約は、テーブル内のどのカラムでも参照できます。create table は 1 つのカラム定義で、複数の検査制約の使用を許可します。

検査制約はカラム定義を上書きしないので、カラム定義が null の使用を許可する場合、null 値の使用を許可しない検査制約を使用することはできません。null 値が許可されるカラムで check 制約を宣言すると、*search_condition* に null が含まれていない場合でも、暗黙的にまたは明示的に、そのカラムに null を挿入できます。たとえば、null 値の使用を許可するテーブル・カラムに次のような検査制約を定義するとします。

```
check (pub_id in ("1389", "0736", "0877", "1622", "1756"))
```

この場合は、そのカラムに null を挿入できます。次の式は常に true に評価されるため、カラム定義は検査制約を上書きします。

```
col_name != null
```

参照整合性を使用するアプリケーションの設計

参照整合性の機能を使用するアプリケーションを設計するときは、以下に従います。

- 不要な参照整合性を作成しないでください。テーブルの参照整合性が増えるほど、そのテーブルでの参照整合性を必要とする文の実行が遅くなります。
- テーブルの自己参照制約は、できるだけ少なくしてください。
- 限定された、特定の範囲の値を検査するアプリケーションには、references 制約ではなく、check 制約を使用してください。check 制約を使用することによって、参照がなくなり、Adaptive Server がクエリを完了するために他のテーブルをスキャンする必要性をなくすことができます。これにより、このようなテーブルのクエリは、参照を使用する他のテーブルよりも速く実行できます。

  たとえば、次のテーブルは、check 制約を使用して、カリフォルニア州に住む作家に限定します。

  ```
  create table cal_authors
  (au_id id not null,
  au_lname varchar(40) not null,
  au_fname varchar(20) not null,
  phone char(12) null,
  address varchar(40) null,
  city varchar(20) null,
  state char(2) null
       check(state = "CA"),
  country varchar(12) null,
  postalcode char(10) null)
  ```

- パフォーマンスを最適化するために、頻繁にスキャンされる外部キー・インデックスを、それ自身のキャッシュにバインドします。ユニーク・インデックスは、プライマリ・キー・カラムに自動的に作成されます。通常、このようなインデックスは、対応する外部キーが更新または挿入されたときに、参照先テーブルをスキャンするために選択されます。
- 候補キーの複数のローの更新は、最小限に抑えてください。
- 参照整合性クエリは、制約検査を使用するプロシージャに挿入してください。制約検査は実行プランにコンパイルされます。参照制約が変更されると、コンパイルされた制約を持つプロシージャは、実行されるときに自動的に再コンパイルされます。
- 参照整合性クエリをプロシージャに埋め込むことができず、参照整合性クエリを特定のバッチで頻繁に再コンパイルする必要がある場合は、システム・カタログ sysreferences を、それ自身のキャッシュにバインドします。これによって、Adaptive Server が参照整合性クエリを再コンパイルするときのパフォーマンスが向上します。

- 参照制約を持つテーブルをテストするには、そのテーブルを使用してクエリを実行する前に、set showplan, noexec on を使用します。showplan 出力は、クエリの実行に必要な補助スキャン記述子の数を示します。スキャン記述子は、テーブルでクエリが実行されるときに、テーブルのスキャンを管理します。補助スキャン記述子の数が非常に多い場合は、テーブルを再設計して使用するスキャン記述子の数を少なくするか、または number of auxiliary scan descriptors 設定パラメータの値を増加します。

テーブルの設計と作成の方法

この項では、ユーザが自分の練習用テーブルを作成するために使用できる create table 文を示します。create table パーミッションを持っていない場合は、システム管理者または作業しているデータベースの所有者に連絡してください。

テーブルを作成し、データを入力してそのテーブルでしばらく作業を行った後で、インデックス、デフォルト、ルール、トリガ、またはビューを作成できます。これによって、どのようなトランザクションが最も多く、どのようなデータが頻繁に入力されるのかを調べることができます。

しかし、多くの場合、より効率的な方法は、テーブルおよびそれに伴う他のすべてのコンポーネントを一度に設計することです。テーブルとそのコンポーネントを実際に作成する前に、プランを紙に描き出してみるとよいでしょう。

まず、次のように、テーブル設計のプランを立てます。

1　テーブルで必要なカラム、そしてそれぞれのデータ型、長さ、精度、および位取りを決定します。

2　ユーザ定義データ型は、それらを使用するテーブルを定義する前に作成します。

3　カラムがある場合は、どのカラムを IDENTITY カラムにするかを決定します。

4　どのカラムに null 入力を許可し、どのカラムに許可しないかを決定します。

5　どの整合性制約、またはカラム・デフォルトがある場合はどのカラム・デフォルトを、テーブルのカラムに追加する必要があるかを決定します。これには、デフォルト、ルール、インデックス、およびトリガの代わりに、カラム制約およびカラム・デフォルトをいつ使用するかの決定も含まれます。

6　デフォルトとルールが必要かどうか、必要な場合は使用する場所と種類を決定します。カラムの null および not null ステータスと、デフォルトおよびルールの間の関係も考慮してください。

7　どのインデックスがどこで必要かを決定します。「第 13 章 テーブルのインデックスの作成」を参照してください。

テーブルとそれに関連するオブジェクトを作成します。

1 create table および create index を使用して、テーブルとそのインデックスを作成します。

2 create default および create rule を使用して、デフォルトとルールを作成します。「第 14 章 データのデフォルトとルールの定義」を参照してください。

3 sp_bindefault および sp_bindrule を使用して、デフォルトとルールをバインドします。create table 文で使用したユーザ定義データ型にデフォルトまたはルールがあった場合は、それらが自動的に使用されます。「第 17 章 ストアド・プロシージャの使用」を参照してください。

4 create trigger を使用して、トリガを作成します。「第 20 章 トリガ：参照整合性」を参照してください。

5 create view を使用して、ビューを作成します。「第 12 章 ビュー：データへのアクセスの制限」を参照してください。

設計スケッチの作成

この章および後続の章では、friends_etc というテーブルを使用して、インデックス、デフォルト、ルール、トリガなどを作成する方法を示します。このテーブルには、友人の名前、住所、電話番号、および個人情報が保持されます。カラム・デフォルトや整合性制約は定義しません。

別のユーザが既に friends_etc テーブルを作成している場合、例を実行して friends_etc に伴うオブジェクトを作成するのであれば、システム管理者かデータベース所有者に確認してください。friends_etc の所有者は、このテーブルのインデックス、デフォルト、ルール、およびトリガを削除して、これらのオブジェクトを作成するときに競合が発生しないようにしてください。

表 8-4 に、friends_etc テーブルと、各カラムに伴うインデックス、デフォルト、ルールを示します。

表 8-4: サンプル・テーブルの設計

カラム	データ型	null	インデックス	デフォルト	ルール
pname	nm	not null	nmind (複合)		
sname	nm	not null	nmind (複合)		
address	varchar(30)	null			
city	varchar(30)	not null		citydflt	
state	char(2)	not null		statedflt	
zip	char(5)	null	zipind	zipdflt	ziprule
phone	p#	null			phonerule
age	tinyint	null			agerule
bday	datetime	not null		bdflt	
gender	bit	not null		gndrdflt	

カラム	データ型	null	インデックス	デフォルト	ルール
debt	money	not null		gndrdflt	
notes	varchar(255)	null			

ユーザ定義データ型の作成

最初の 2 つのカラムは名と姓です。これは nm データ型で定義されています。テーブルを作成する前に、データ型を作成します。phone カラムの p# データ型にも同じことがあてはまります。

```
execute sp_addtype nm, "varchar(30)"
execute sp_addtype p#, "char(10)"
```

nm データ型は、最大 30 バイトでの可変長の文字入力を許可します。p# データ型は、10 バイトの固定長の char データ型を許可します。

null 値を許可するカラムの選択

ユーザ定義データ型が割り当てられたカラムを除いて、各カラムには明示的な null または not null エントリがあります。テーブル定義で not null を指定する必要はありません。これはデフォルトです。このテーブル設計では、読みやすいように not null を明示的に指定しています。

not null デフォルトは、そのカラムにエントリが必要であることを意味します。たとえば、このテーブルの 2 つの名前用カラムです。名前がなければ、他のデータには意味がありません。さらに、gender カラムも not null でなければなりません。bit カラムでは null を使用できないためです。

null と指定されているカラムにデフォルトがバインドされている場合は、入力時に他の値が入力されないかぎり、null ではなくデフォルト値が入力されます。null と指定されているカラムに、null を指定しないルールがバインドされている場合は、カラムに値が入力されないと、カラム定義がルールを上書きします。カラムはデフォルトとルールの両方を持つことができます。デフォルトとルールの関係については、「第 14 章 データのデフォルトとルールの定義」で説明しています。

テーブルの定義

create table 文を作成します。

```
create table friends_etc
(pname       nm             not null,
sname        nm             not null,
address      varchar(30)    null,
city         varchar(30)    not null,
state        char(2)        not null,
postalcode   char(5)        null,
```

```
    phone       p#              null,
    age         tinyint         null,
    bday        datetime        not null,
    gender      bit             not null,
    debt        money           not null,
    notes       varchar(255)    null)
```

これで個人名および姓、住所、都市、州、郵便番号、電話番号、年齢、生年月日、債務情報、メモのカラムが定義されました。後で、このテーブルのルール、デフォルト、インデックス、トリガ、ビューを作成します。

クエリ結果からの新しいテーブルの作成：*select into*

select into コマンドを使用すると、select 文の select リストで指定されたカラムと where 句で指定されたローに基づいて、新しいテーブルを作成できます。into 句は、テスト・テーブルおよび既存のテーブルのコピーでの新しいテーブルの作成、大きなテーブルからの複数の小さなテーブルの作成に便利です。

select 句および select into 句は、delete 句や update 句と同様に、TOP 機能を有効にします。TOP オプションは、ターゲット・テーブルに挿入されるローの数を制限できるようにする符号なし整数で、他のプラットフォームとの互換性を実現します。『リファレンス・マニュアル：コマンド』を参照してください。

select into/bulkcopy/pllsort データベース・オプションが on に設定されている場合にかぎり、永久テーブルで select into を使用できます。システム管理者は、sp_dboption を使用して、このオプションを on にできます。このオプションが on になっているかどうかを確認するには、sp_helpdb を使用してください。

select into/bulkcopy/pllsort データベース・オプションが on に設定されているときに、sp_helpdb とその結果がどのように表示されるかを次に示します。この例で使用するページ・サイズは 8K です。

```
    sp_helpdb pubs2
```

```
name      db_size    owner     dbid created        status
--------- ---------- --------- ---- -------------- --------------
pubs2     20.0 MB     sa            4 Apr 25, 2005   select
     into/bulkcopy/pllsort, trunc log on chkpt, mixed log and data

device_fragments    size           usage          created      free kbytes
------------------- -------------- -------------- -----------  ------------
master               10.0MB        data and log  Apr 13 2005           1792
pubs_2_dev          10.0MB        data and log  Apr 13 2005          9888

device                  segment
----------------------- ----------------------
master                  default
master                  logsegment
master                  system
```

```
pubs_2_dev              default
pubs_2_dev              logsegment
pubs_2_dev              system
pubs_2_dev              seg1
pubs_2_dev              seg2
```

sp_helpdb の出力は、オプションが on または off のどちらに設定されているかを示します。

select into/bulkcopy/pllsort データベース・オプションが on に設定されている場合は、select into 句を使用して、create table 文を使用することなく新しい永久テーブルを構築できます。select into/bulkcopy/pllsort オプションが on に設定されていなくても、テンポラリ・テーブルに select into を実行することができます。

注意　select into は最小限にしかログに記録されない演算なので、select into の後に dump database を使用して、データベースをバックアップしてください。最小限にしかログに記録されない演算に続けてトランザクション・ログをダンプすることはできません。

テーブルの一部を表示するビューとは異なり、select into で作成したテーブルは、個別の独立したエンティティです。「第 12 章 ビュー：データへのアクセスの制限」を参照してください。

新しいテーブルは、select リストで指定したカラム、from 句で指定したテーブル、および where 句で指定したローに基づいています。新しいテーブルの名前はデータベース内でユニークであり、識別子の規則に従っている必要があります。

into 句を指定した select 文を使用すると、通常のデータ定義プロセスを実行することなく、既存の定義とデータに基づいてテーブルを定義してデータを挿入することができます。

次の例は、select into 文とその結果を示します。この例では、4 つのカラムを持つテーブル publishers のうち、2 つのカラムを使用して、newtable というテーブルを作成します。この文には where 句が含まれていないため、publishers のすべてのロー (ただし、指定された 2 つのカラムの分のみ) のデータが newtable にコピーされます。

```
select pub_id, pub_name
into newtable
from publishers
(3 rows affected)
```

“3 rows affected” は、newtable に 3 つのローが挿入されたことを示します。newtable は次のようになります。

```
select *
from newtable
```

```
pub_id  pub_name
------  ------------------------------------
0736    New Age Books
0877    Binnet & Hardley
1389    Algodata Infosystems
```

新しいテーブルには select 文の結果が含まれます。これは、親テーブルと同様、データベースの一部となります。

where 句に偽の条件を挿入することによって、データを持たない空のテーブルを作成できます。次に例を示します。

```
select *
into newtable2
from publishers
where 1=2

(0 rows affected)

select *
from newtable2

pub_id     pub_name         city      state
-------    --------------    --------  ------
```

1 = 2 となることは決してないので、新しいテーブルにはローは挿入されません。

また、select into を集合関数とともに使用して、計算データを持つテーブルを作成することもできます。

```
select type, "Total_amount" = sum(advance)
into #whatspent
from titles
group by type

(6 rows affected)

select * from #whatspent

 type          Total_amount
 ------------ ------------------------
UNDECIDED                        NULL
business                    25,125.00
mod_cook                    15,000.00
popular_comp                15,000.00
psychology                    21,275.00
trad_cook                   19,000.00
```

集合関数またはその他の式から結果として生成される select into 結果テーブルのカラムには、必ず名前を提供してください。例を示します。

- 算術集合。たとえば *amount* * 2。
- 連結。たとえばlname + fname。
- 関数。たとえば、lower(lname)

次の例は連結を使用します。

```
select au_id,
    "Full_Name" = au_fname + ' ' + au_lname
into #g_authortemp
from authors
where au_lname like "G%"

(3 rows affected)

select * from #g_authortemp

 au_id       Full_Name
 ----------- -------------------------
 213-46-8915 Marjorie Green
 472-27-2349 Burt Gringlesby
 527-72-3246 Morningstar Greene
```

関数は null 値の使用を許可するので、convert または isnull 以外の関数の結果として生成されるテーブルのカラムは、いずれも null 値の使用を許可します。

エラーのチェック

select into は 2 段階の演算です。第 1 段階では新しいテーブルを作成し、第 2 段階では指定されたローをテーブルに挿入します。

select into 演算はログに記録されないので、ユーザ定義のトランザクション内では発行できず、ロールバックできません。

新しいテーブルの作成後に select into 文が失敗した場合、Adaptive Server は、テーブルを自動的に削除したり、最初のデータ・ページの割り付けを自動的に解除したりしません。これは、エラーが発生する前に最初のページに挿入されたローがそのページに残ることを意味します。select into 文の後で *@@error* グローバル変数の値を調べて、エラーが発生していないことを確認してください。

select into 演算でエラーが発生した場合は、drop table を使用して新しいテーブルを削除し、select into 文を再発行してください。

IDENTITY カラムとの *select into* の使用

この項では、IDENTITY カラムを持つテーブルとの select into コマンドの使用についての特別な規則を説明します。

新しいテーブルへの IDENTITY カラムの選択

新しいテーブルに既存の IDENTITY カラムを選択するには、次のように、select 文の *column_list* にカラム名 (または syb_identity キーワード) を含めます。

```
select column_list
    into table_name
    from table_name
```

次の例では、stores_cal テーブルからの結果に基づいて、新しいテーブル stores_cal_pay30 を作成します。

```
select record_id, stor_id, stor_name
into stores_cal_pay30
from stores_cal
where payterms = "Net 30"
```

次に挙げる条件のいずれかがあてはまらない場合、新しいカラムは IDENTITY プロパティを継承します。

- IDENTITY カラムが複数回選択されている。
- IDENTITY カラムが式の一部として選択されている。
- select 文に group by 句、集合関数、union 演算子、またはジョインが含まれている。

IDENTITY カラムの複数回の選択

テーブルは複数の IDENTITY カラムを持つことはできません。IDENTITY カラムは、複数回選択されると、新しいテーブル内で not null として定義されます。IDENTITY プロパティは継承されません。

次の例では、record_id カラムが名前で一度、そして syb_identity キーワードで一度選択されています。このカラムは、stores_cal_pay60 内で not null として定義されます。

```
select syb_identity, record_id, stor_id, stor_name
into stores_cal_pay60
from stores_cal
where payterms = "Net 60"
```

select into による新しい IDENTITY カラムの追加

select into 文で新しい IDENTITY カラムを定義するには、into 句の前にカラム定義を追加します。定義にはカラムの精度が含まれますが、位取りは含まれません。

```
select column_list
    identity_column_name = identity(precision)
  into table_name
  from table_name
```

次の例では、discounts テーブルから新しいテーブル new_discounts を作成し、新しい IDENTITY カラム id_col を追加します。

```
select *, id_col=identity(5)
into new_discounts
from discounts
```

column_list に既存の IDENTITY カラムが含まれている場合、新しい IDENTITY カラムの記述を追加すると、select into 文は失敗します。

値を計算する必要があるカラムの定義

IDENTITY カラムの値は Adaptive Server によって生成されます。IDENTITY カラムに基づく新しいカラムの値が、生成されるのではなく計算される必要がある場合、新しいカラムは IDENTITY プロパティを継承できません。

テーブルの select 文に IDENTITY カラムが式の一部として含まれる場合、結果のカラム値は計算される必要があります。式中のいずれかのカラムが null 入力可である場合、新しいカラムは null として作成されます。それ以外の場合は not null になります。

次の例では、record_id の値に 1000 を加えることによって計算される new_id カラムが、not null として作成されます。

```
select new_id = record_id + 1000, stor_name
into new_stores
from stores_cal
```

select 文に group by 句または集合関数が含まれる場合にも、カラム値が計算されます。IDENTITY カラムが集合関数の引数である場合、結果のカラムは null として作成されます。それ以外の場合は not null になります。

union またはジョインによってテーブルに選択される IDENTITY カラム

IDENTITY カラムには、請求書番号や従業員番号など Adaptive Server が自動的に生成するユニークな番号が保管されます。しかし、テーブルの select 文に union またはジョインが含まれている場合、個々のローは結果セットに複数回表示されることがあります。union やジョインによってテーブルに選択される IDENTITY カラムは、IDENTITY プロパティを保持しません。テーブルに IDENTITY カラムの union および null カラムが含まれる場合、新しいカラムは null として定義されます。それ以外の場合は not null になります。

詳細については、「IDENTITY カラムの使用」(269 ページ)、「IDENTITY カラムの更新」(235 ページ)、および『リファレンス・マニュアル：コマンド』を参照してください。

既存のテーブルの変更

既存のテーブルの構造を変更するには、alter table コマンドを使用します。次の処理ができます。

- カラムと制約の追加
- カラムのデフォルト値の変更
- null カラムと not null カラムの追加
- カラムと制約の削除

- ロック・スキームの変更
- テーブルの分割または分割解除
- カラムのデータ型の変換
- 既存のカラムの null デフォルト値の変換
- カラム長の拡張または短縮

テーブルのパーティション属性を変更することもできます。詳細については、「第 10 章 テーブルとインデックスの分割」および『リファレンス・マニュアル：コマンド』を参照してください。

たとえば、デフォルトでは authors テーブルの au_lname カラムは、varchar(50) データ型を使用します。varchar(60) を使用するよう au_lname を変更するには、次のように入力します。

```
alter table authors
modify au_lname varchar(60)
```

注意 alter table 文に含まれるデフォルトに対して、変数を引数として使用することはできません。

非 null のカラムの削除、修正、および追加では、データ・コピーを実行する場合があります。これは、必要な領域とロック・スキームにも関わります。「データ・コピー」(312 ページ) を参照してください。

修正したテーブルのページ・チェーンは、そのテーブルの現在の設定オプションを継承します (たとえば、fillfactor が 50 パーセントに設定されていれば、新しいページの fillfactor も同じになります)。

注意 Adaptive Server は、alter table オペレーションに対して (ページ割り付けの) 部分ロギングを実行します。ただし、alter table は 1 つのトランザクションとして実行されるので、alter table を実行した後でトランザクション・ログをダンプすることはできません。データベースをダンプして、リカバリ可能であることを確認してください。alter table オペレーション中にサーバで問題が発生した場合、Adaptive Server はトランザクションをロールバックします。

---

alter table は、テーブル・スキーマを修正している間、排他テーブル・ロックを取得します。このロックは、コマンドが終了すると同時に解放されます。

alter table はトリガを起動しません。

テーブルへの変更をリストしない、*select* * を使用するオブジェクト

カラムを削除したテーブル上で select * を実行するオブジェクト (ストアド・プロシージャ、トリガなど) がデータベースにある場合は、エラー・メッセージが表示され、足りないカラムがリストされます。これは、with recompile オプションを使用してオブジェクトを作成している場合にも発生します。たとえば、authors テーブルから postalcode カラムを削除した場合、このテーブル上で select * を実行したストアド・プロシージャは、次のエラー・メッセージを表示します。

```
Msg 207, Level 16, State 4:
Procedure 'columns', Line 2:
Invalid column name 'postalcode'.
(return status = -6)
```

新しいカラムを追加してから select * を含むオブジェクトを実行した場合は、このメッセージは表示されません。この場合、新しいカラムは出力に表示されません。

削除されたカラムを参照するオブジェクトは、削除して再作成してください。

リモート・テーブルでの *alter table* の使用

alter table を使用して、コンポーネント統合サービス (CIS) を使用するリモート・テーブルを修正できます。リモート・テーブルを修正する前に、次を入力して CIS が稼働しているか確認します。

```
sp_configure "enable cis"
```

CIS が有効になっている場合、このコマンドの出力は “1” です。CIS は、Adaptive Server のインストール時にデフォルトで有効になります。

『システム管理ガイド：第 1 巻』の「第 5 章 設定パラメータ」および『コンポーネント統合サービス・ユーザーズ・ガイド』を参照してください。

カラムの追加

次の文は、デフォルト値として定数 “primary_author” を含む、author_type という非 null カラムと、au_publisher という null カラムを、authors テーブルに追加します。

```
alter table authors
add author_type varchar(20)
default "primary_author" not null,
au_publisher varchar(40) null
```

カラムの追加によるカラム ID の付加

alter table は、現在の最大カラム ID より 1 つ大きなカラム ID を持つカラムを、テーブルに追加します。たとえば、表 8-5 は、salesdetail テーブルのデフォルトのカラム ID のリストです。

表 8-5: salesdetail テーブルのカラム ID

カラム名	stor_id	ord_num	title_id	qty	discount
カラム ID	1	2	3	4	5

次のコマンドは、store_name カラムを、6 というカラム ID で salesdetail テーブルの最後に付加します。

```
alter table salesdetail
add store_name varchar(40)
default
"unknown" not null
```

もう 1 つカラムを追加すると、そのカラムの ID は 7 になります。

注意 テーブルのカラム ID は、カラムが追加されたり削除されたりすると変わるので、アプリケーションをカラム ID に依存させないでください。

not null カラムの追加

テーブルに not null カラムを追加できます。これは、カラムが追加されると、定数式および null でない値が、カラムに置かれることを意味します。またこれによって、テーブルの作成時に、すべての既存のローについて、新しいカラムに指定の定数式が移植されることを保証します。

ユーザが not null カラムに値を入力しなかった場合、Adaptive Server はエラー・メッセージを返します。

次の文は、カラム owner を、“unknown” というデフォルト値で stores テーブルに追加します。

```
alter table stores
add owner_lname varchar(20)
default "unknown" not null
```

null カラムを追加するときは、デフォルト値は定数式でもかまいませんが、(前述の例のように) not null カラムを追加するときは、定数値のみ可能です。

制約の追加

既存のカラムに制約を追加するには、alter table を使用します。たとえば、前払い金が 10,000 を超えないようにする制約を titles テーブルに追加するには、次のように入力します。

```
alter table titles
```

```
add constraint advance_chk
check (advance < 10000)
```

ユーザが 10,000 より大きな値を titles テーブルに挿入しようとすると、Adaptive Server は次のようなエラー・メッセージを生成します。

```
Msg 548, Level 16, State 1:
Line 1:Check constraint violation occurred,
dbname = 'pubs2',table name= 'titles',
constraint name = 'advance_chk'.
Command has been aborted.
```

制約の追加は既存のデータに影響しません。また、デフォルト値を持つ新しいカラムを追加して、そのカラムに制約を指定すると、デフォルト値は制約に対して検証されません。

制約の削除については、「制約の削除」(306 ページ) を参照してください。

カラムの削除

既存のテーブルからカラムを削除するには、alter table を使用します。1 つの alter table 文で、カラムをいくつでも削除できます。ただし、最後に残ったカラムをテーブルから削除することはできません (たとえば、5 つのカラムを持つテーブルから 4 つのカラムを削除したら、残りの 1 つは削除できません)。

たとえば、次の文は、titles テーブルから advance カラムと contract カラムを削除します。

```
alter table titles
drop advance, contract
```

alter table は、カラムを削除するときに、テーブルにあるインデックスをすべて再構築します。

カラムの削除によるカラム ID の再番号付け

テーブルからカラムを削除するときに、alter table はカラム ID を再番号付けします。削除されたカラムの番号より大きな ID を持つカラムは、ID が 1 つ繰り下がり、削除されたカラムによって発生したギャップを埋めます。たとえば、titleauthor テーブルには、次のカラム名およびカラム ID が含まれています。

表 8-6: titleauthor のカラム ID

カラム名	au_id	title_id	au_ord	royaltyper
カラム ID	1	2	3	4

テーブルから au_ord カラムを削除する場合は、次のように入力します。

```
alter table titleauthor drop au_ord
```

その結果、titleauthor テーブルに含まれているカラム名およびカラム ID は次のようになります。

表 8-7: au_ord を削除した後のカラム ID

カラム名	au_id	title_id	royaltyper
カラム ID	1	2	3

royaltyper カラムのカラム ID は 3 になります。title_id と royaltyper の両方のノンクラスタード・インデックスも、au_ord が削除されたときに再構築されます。また、異なるシステム・カタログ内のカラム ID のすべてのインスタンスも、再番号付けされます。

ユーザは、通常はカラム ID の再番号付けには気付きません。

注意 テーブルのカラム ID は、カラムが追加または削除されると再番号付けされるので、アプリケーションをカラム ID に依存させないでください。カラム ID に依存するストアド・プロシージャやアプリケーションがある場合は、適切なカラム ID にアクセスするように、ストアド・プロシージャやアプリケーションを作成し直します。

制約の削除

制約を削除するには、alter table を使用します。次に例を示します。

```
alter table titles
drop constaint advance_chk
```

「sp_helpconstraint によるテーブルの制約情報の表示」(332 ページ) を参照してください。

カラムの修正

alter table を使用して、既存のカラムを修正します。単独の alter table 文で、カラムをいくつでも修正できます。

たとえば、次のコマンドは、titles テーブルの type カラムのデータ型を char(12) から varchar(20) に変更し、null 入力可能にします。

```
alter table titles
modify type varchar(20) null
```

警告！ 持っているオブジェクトの中に、特定のデータ型のカラムに依存するもの (ストアド・プロシージャ、トリガなど) が含まれる場合があります。カラムを修正する前に、テーブルの従属オブジェクトを判断し、これらのオブジェクトを参照するオブジェクトが、修正後に正常に実行できることを確認するために、sp_depends を使用します。

データ型の変換

変換は、暗黙的または明示的に新しいデータ型に変換できるデータ型に対してのみ可能です。または、明示的な変換関数が Transact-SQL にある場合に限ります。サポートされているデータ型変換のリストについては、『リファレンス・マニュアル：ビルディング・ブロック』の「第 1 章 システム・データ型とユーザ定義データ型」を参照してください。不正なデータ型修正を実行しようとすると、Adaptive Server はエラー・メッセージを返し、オペレーションはアボートされます。

注意 既存のカラムのデータ型を timestamp データ型に変換したり、timestamp データ型を使用するカラムを他のデータ型に修正したりすることはできません。

同一の alter table...modify コマンドを 2 回以上発行すると、Adaptive Server は、次のようなメッセージを返します。

```
Msg 13905, Level 16, State 1:
Server 'SYBASE1', Line 1:
Warning: no columns to drop, add or modify. ALTER TABLE 'authors' was aborted.
```

テーブルの修正による、以前のダンプのバルク・コピーの失敗

カラムの長さまたはデータ型を修正すると、テーブルの以前のダンプのバルク・コピー・インを正常に実行できなくなる場合があります。以前のテーブル・スキーマは、新しいテーブル・スキーマと互換でない可能性があります。カラムの長さまたはデータ型を修正する前に、その修正によってテーブルの以前のダンプのコピー・インが妨げられることがないことを確認してください。

カラム長の短縮によるデータのトランケート

カラムの長さを短縮する場合は、短縮されたカラム長によってデータのトランケートが発生しないことを確認してください。たとえば alter table を使用して、titles テーブルの title カラムの長さを varchar(80) から varchar(2) に短縮することができますが、そうすると、データは意味のないものになります。

```
select title from titles

title
-----
Bu
Co
Co
Em
Fi
Is
Li
Ne
```

```
On
Pr
Se
Si
St
Su
Th
Th
Th
Yo
```

Adaptive Server は、set string_rtruncation オプションが on になっている場合にかぎり、カラム・データのトランケートについてエラー・メッセージを返します。文字データをトランケートする必要がある場合は、適切な string-truncation オプションを設定し、カラムを修正してカラム長を短縮します。

datetime カラムの修正

カラムを char データ型から datetime、smalldatetime、または date に修正する場合は、出力に表示する月、日、年の順序を指定できません。また、出力に使用する言語も指定できません。これらの設定はいずれもデフォルト値が使用されます。ただし、set dateformat または set language を使用して、カラムに格納された情報の設定に合わせて出力を変更できます。また、Adaptive Server は、smalldatetime から char データ型へのカラムの修正をサポートしていません。『リファレンス・マニュアル：コマンド』を参照してください。

カラムの null デフォルト値の修正

カラムの null デフォルト値だけを変更する場合、カラムのデータ型を指定する必要はありません。たとえば、次のコマンドは、authors テーブルの address カラムを null から not null に変更します。

```
alter table authors
modify address not null
```

カラムを修正し、データ型を not null と指定した場合、null 値を持つローがないかぎり、オペレーションは成功します。しかし、いずれかのローに null 値が含まれていると、オペレーションは失敗し、不完全なトランザクションはロールバックされます。たとえば次の文は、titles テーブルの『The Psychology of Computer Cooking』に null 値が含まれているため、失敗します。

```
alter table titles
modify advance numeric(15,5) not null

Attempt to insert NULL value into column 'advance', table
'pubs2.dbo.titles';
column does not allow nulls.Update fails.
Command has been aborted.
```

このコマンドを正常に実行するには、修正されたカラムの null 値がすべて not null に変更されるようにテーブルを更新し、その後、コマンドを再発行します。

精度または位取りを持つカラムの修正

カラムの位取りを修正する前に、データの長さを確認します。

alter table コマンドによってカラム値の精度が低くなる場合 (たとえば numeric(10,5) から numeric(5,5)) など)、Adaptive Server はその文をアボートします。文がバッチの一部である場合、arithabort arithignore arith_overflow オプションが on になっているとそのバッチがアボートされます。

alter table コマンドによってカラム値の位取りが低くなる場合 (たとえば numeric(10, 5) から numeric(10,3) など) は、警告が出されることなくローがトランケートされます。これは、arithabort numeric_truncation が on であるか off であるかに関係なく発生します。

arithignore arith_overflow が on に設定されている場合、alter table によって数値オーバフローが発生すると、Adaptive Server は警告を発行します。しかし、arithabort arithignore arith_overflow が off に設定されている場合は、alter table によって数値オーバフローが発生しても、Adaptive Server は警告を発行しません。デフォルトでは、Adaptive Server のインストール時に、arithignore arith_overflow は off に設定されます。

注意　カラムの長さをトランケートする可能性のあるコマンドを運用環境で発行する場合は、その前にデータ・トランケーションの規則を参照し、その意味を完全に理解していることを確認してください。最初はテスト・カラムのセットでコマンドを実行してください。

---

text カラム、*unitext* カラム、*image* カラムの修正

text カラムは、次のものに変換できます。

- [n]char
- [n]varchar
- unichar
- univarchar

unitext カラムは、次のものに変換できます。

- [n]char
- [n]varchar
- unichar
- univarchar
- binary
- varbinary

image カラムは、次のものに変換できます。

- varbinary
- binary

char、varchar、unichar、univarchar データ型のカラムを text カラムまたは unitext カラムに修正することはできません。text または unitext から char、varchar、unichar、univarchar に変換している場合、カラムの最大長はページ・サイズによって決まります。カラム長を指定しないと、alter table はデフォルトの長さである 1 バイトを使用します。マルチバイト文字の text、unitext、image カラムを修正する場合、データを収容するのに十分なカラム長を指定しないと、Adaptive Server はカラム長に合わせてデータをトランケートします。

IDENTITY カラムの追加、削除、および修正

この項では、alter table を使用した IDENTITY カラムの追加、削除、および修正について説明します。IDENTITY カラムの詳細については、「IDENTITY カラムの使用」(269 ページ)を参照してください。

IDENTITY カラムの追加

not null のデフォルト値が指定されている場合のみ、IDENTITY カラムを追加できます。新しい IDENTITY カラムにデフォルト句を指定することはできません。

テーブルに IDENTITY カラムを追加するには、alter table 文に identity キーワードを指定します。

```
alter table table_name add column_name
    numeric(precision ,0) identity not null
```

次の例では、IDENTITY カラム record_id を stores テーブルに追加します。

```
alter table stores
    add record_id numeric(5,0) identity not null
```

テーブルに IDENTITY カラムを追加すると、Adaptive Server は、値 1 から始まる、ユニークな連続する値をそれぞれのローに割り当てます。テーブルに含まれるローの数が多い場合、このプロセスには時間がかかります。ローの数がカラムの最大値を超えていると (この場合は 10^5 - 1、つまり 99,999)、alter table 文は失敗します。

ユーザ定義データ型を使用して IDENTITY カラムを作成できます。ユーザ定義データ型は、位取りが 0 の numeric 型である必要があります。

IDENTITY カラムの削除

IDENTITY カラムは、他のカラムと同じように削除できます。次に例を示します。

```
alter table stores
drop record_id
```

IDENTITY カラムの削除には以下の制限があります。

- データベースの sp_dboption “identity in nonunique index” が on になっている場合、最初にすべてのインデックスを削除し、次に IDENTITY カラムを削除します。それからインデックスを再作成します。

  IDENTITY カラムが隠しカラムである場合は、まず syb_identity キーワードを使用してカラムを識別します。「syb_identity による IDENTITY カラムの参照」(271 ページ) を参照してください。

- set identity_insert が on になっているテーブルから IDENTITY カラムを削除するには、まず sp_helpdb を発行して、set identity_insert が on になっているかどうかを確認します。

  次に、set identity_insert オプションをオフにします。

  ```
  set identity_insert table_name off
  ```

  IDENTITY カラムを削除し、新しい IDENTITY カラムを追加します。その後、set identity_insert オプションを on にします。

  ```
  set identity_insert table_name on
  ```

IDENTITY カラムの修正

IDENTITY カラムのサイズを修正して、範囲を拡大することができます。これは、現在の範囲が小さすぎる場合や、サーバの停止によって範囲が使い果たされてしまった場合に必要になります。

たとえば、次のように入力して、record_id の範囲を拡大できます。

```
alter table stores
modify record_id numeric(9,0)
```

範囲を縮小するには、ターゲットのデータ型に現在より小さい精度を指定します。テーブルの IDENTITY 値がターゲット IDENTITY カラムの範囲には大きすぎる場合、算術変換が行われて alter table は文をアボートします。

分割されたテーブルに非 null の IDENTITY カラムを追加するために、データ・コピーを必要とする alter table コマンドを使用することはできません。分割されたテーブルに対するデータ・コピーは並列に行われるため、ユニークな IDENTITY 値は保証できません。

データ・コピー

Adaptive Server は、テーブル・スキーマを変更する前にテーブルからデータを一時的にコピーする必要がある場合だけ、データ・コピーを実行します。テーブルにインデックスがある場合、Adaptive Server は、データ・コピーの終了後にインデックスを再構築します。

注意 alter table コマンドがデータ・コピーを実行している場合、そのテーブルを含むデータベースでは、select into/bulkcopy/pllsort が on に設定されている必要があります。『リファレンス・マニュアル：コマンド』を参照してください。

Adaptive Server は、次の場合にデータ・コピーを実行します。

- カラムを削除するとき。
- 次に示すカラムのプロパティを修正するとき。
    - データ型 (varchar、varbinary、null char、null binary のカラム長を拡張する場合を除く)。
    - null から not null へ (またはその逆)。
    - 長さ短縮。カラム長を短縮する場合、短縮したカラム長にすべてのデータが収納できるかどうか事前にわからない場合がある。たとえば、au_lname を varchar(30) に短縮する場合に、varchar(35) を必要とする名前が含まれていることがある。ユーザがカラムのデータ長を短縮しようとすると、Adaptive Server によってまずデータ・コピーが実行され、カラム長の変更に問題がないことが確認される。
- 数値カラムの長さを拡張 (たとえば tinyint を int に) するとき。Adaptive Server は、あるローがこのカラムに対して not null 値を持っていた場合に備えて、データ・コピーを実行する。
- not null カラムを追加するとき。

以下を変更する場合は、alter table はデータ・コピーを実行しません。

- varchar カラム、または varbinary カラムの長さ。
- 物理データ型ではないユーザ定義データ ID。たとえば使用するサイトに、ユーザ定義型は異なるが同じ物理データ型を持つ mychar1 と mychar2 の 2 つのデータ型がある場合、mychar1 を mychar2 に変更してもデータ・コピーは実行されません。
- 可変長カラムの null デフォルト値を not null から null へ変更するとき。

alter table がデータ・コピーを実行するかどうかを識別するには、次の手順に従います。

1 Adaptive Server がデータ・コピーを実行するかどうかをレポートさせるために、showplan を on に設定します。

2　作業が何も実行されないように、noexec を on に設定します。

3　alter table コマンドを実行します。データ・コピーの必要がない場合は、alter table コマンドによる変更を反映するように、カタログ更新だけが実行されます。

exp_row_size の変更

データ・コピーを実行する場合は、ローごとに使用可能な領域を指定できる exp_row_size を変更することもできます。修正されるテーブル・スキーマに可変長カラムが含まれる場合にかぎり、sysindexes にある maxlen および minlen 値が修正されるテーブル・スキーマに対して指定する範囲内にだけ、exp_row_size を変更できます。

カラムに固定長カラムが指定されている場合、exp_row_size は 0 か 1 にしか変更できません。テーブルから可変長カラムをすべて削除した場合は、0 または 1 の exp_row_size を指定する必要があります。また、alter table コマンドで exp_row_size を提供しなかった場合は、以前の exp_row_size が使用されます。テーブルに固定長カラムしか含まれておらず、以前の exp_row_size が修正後のスキーマと互換性がない場合、Adaptive Server はエラーを返します。

exp_row_size 句を、他の alter table サブ句 (たとえば定数の定義、ロック・スキームの変更など) とともに使用することはできません。また、sp_chgattribute を使用して、exp_row_size を変更することもできます。『リファレンス・マニュアル：コマンド』を参照してください。

ロック・スキームとテーブル・スキーマの修正

alter table でデータ・コピーを実行する場合は、テーブルのロック・スキームを変更するコマンドを含めることもできます。たとえば、次の文は、authors テーブルの au_lname カラムを修正し、そのテーブルのロック・スキームを全ページ・ロックからデータロー・ロックに変更します。

```
alter table authors
modify au_lname varchar(10)
lock datarows
```

ただし、alter table を使用して、クラスタード・インデックスが指定されているテーブルのテーブル・スキーマおよびロック・スキームを変更することはできません。テーブルにクラスタード・インデックスが指定されている場合は、次の手順を実行できます。

1　インデックスを削除します。

2　(テーブル・スキーマの変更にもデータ・コピーが含まれる場合) 同一文内で、テーブル・スキーマを修正し、ロック・スキームを変更します。

3　クラスタード・インデックスを再構築します。

または、alter table コマンドを発行してロック・スキームを変更してから、もう一度 alter table コマンドを発行してテーブルのスキーマを変更します。

ユーザ定義データ型を使用するカラムの変更

alter table を使用して、ユーザ定義データ型を使用するカラムを追加、削除、または修正できます。

ユーザ定義データ型を使用するカラムの追加

ユーザ定義データ型のカラムを追加するには、システム定義データ型のカラムの場合と同じ構文を使用します。たとえば、次の文は、usertype を使用して、pubs2 の authors テーブルにカラムを追加します。

```
alter table titles
add newcolumn usertype not null
```

ユーザがデフォルトとして指定する null または not null は、ユーザ定義データ型によって指定されるデフォルトよりも優先されます。つまり、新しいカラムを追加し、デフォルトとして not null を指定すると、そのカラムのデフォルトは、ユーザ定義データ型で null が指定されていても、not null になります。null または not null を指定しないと、ユーザ定義データ型によって指定されるデフォルトが使用されます。

not null のカラムを追加するときは、ユーザ定義データ型にデフォルトがバインドされている場合を除き、default 句を指定してください。

ユーザ定義データ型が IDENTITY カラムのプロパティ (精度と位取り) を指定する場合、カラムは IDENTITY カラムとして追加されます。

ユーザ定義データ型を使用するカラムの削除

ユーザ定義データ型のカラムを削除する方法は、システム定義データ型のカラムを削除する場合と同様です。

ユーザ定義データ型を使用するカラムの修正

ユーザ定義データ型を含むようにカラムを修正する構文は、システム定義データ型を含むようにカラムを修正する構文と同じです。たとえば、次の文は、authors テーブルの au_lname を、ユーザ定義の newtype データ型を使用するように修正します。

```
alter table authors
modify au_lname newtype(60) not null
```

デフォルトとして null または not null を指定しないと、ユーザ定義データ型によって指定されるデフォルトが使用されます。

テーブルを修正しても、カラムにバインドされている現在のルールやデフォルトは影響を受けません。しかし、新しいルールやデフォルトを指定すると、ユーザ定義データ型にバインドされている以前のルールやデフォルトは、削除されます。これまでにカラムにバインドされているルールやデフォルトがない場合、任意のユーザ定義のルールまたはデフォルトが適用されます。

既存のカラムを IDENTITY カラムに変更することはできません。修正できる既存の IDENTITY カラムは、IDENTITY カラムのプロパティ (精度と位取り) を持つユーザ定義データ型のものだけです。

alter table からのエラーと警告

alter table の実行時に発生するエラーのほとんどは、要求したコマンドの実行を妨げるスキーマ構成体をユーザに通知するものです (たとえば、インデックスの一部であるカラムを削除しようとした場合など)。影響を受けるカラムに依存するスキーマ・オブジェクトのエラーまたは警告を解決してから、コマンドを再発行してください。エラー状態をレポートさせるには、次の手順に従います。

1　showplan を on に設定します。

2　noexec を on に設定します。

3　alter table コマンドを実行します。

レポートされたエラーを処理するようにコマンドを変更したら、Adaptive Server が実際に作業を実行できるように、showplan および noexec を off に設定します。

alter table は、実際にコマンドを実行しているときに特定のエラーを検出し、レポートします (たとえばカラムを削除している場合の参照制約の存在など)。ランタイムのデータに依存するエラー (たとえば、数値オーバフロー、文字トランケーションなどのエラー) はすべて、その文が実行されたときだけ識別されます。使用可能なデータに合わせてコマンドを変更するか、その文で指定されるターゲットのデータ型で作用するようにデータ値を修正する必要があります。このようなエラーを識別するには、noexec を無効にして、コマンドを実行します。

alter table modify によって生成されるエラーと警告

alter table modify コマンドによってのみ生成されるエラーがあります。alter table modify は互換性のあるデータ型にカラムを変換しますが、変換するカラムに特定の制限があると、alter table がエラーを発行する場合があります。

注意 コマンドを発行する前に、データ型を修正することの影響を理解していることを確認してください。一般的に、alter table modify は、変換可能なデータ型間での変換を暗黙的に実行する場合にだけ使用します。これを使用することによって、insert および update 文の処理中に必要な隠し変換を、データ型間の互換性がないために失敗することなく実行できます。

たとえば、カラム second_advance を、データ型として int を指定して titles テーブルに追加し、second_advance カラムにクラスタード・インデックスを作成すると、このカラムを char データ型に修正できなくなります。これによって int 値は、整数 (1、2、3) から文字列 (1、2、3) に変換されます。インデックスが格納されたデータで再構築されるときに、データ値は格納された順序であることが期待されます。しかし、この例では、データ型は int から char に変更されており、char データ型の順序付けシーケンスによる順序ではありません。このため、alter table コマンドは、インデックスの再構築段階で失敗します。

クラスタード・インデックスのインデックス・キー・カラムの一部であるカラムに新しいデータ型を選択するときは、注意が必要です。alter table modify では、データがコピーされた後で、修正されるデータの順序付けシーケンスに違反しないデータ型を指定する必要があります。

alter table modify は、制約を含むカラム内でユーザがデータ型を、互換性のないデータ型に修正した場合も、警告メッセージを発行します。たとえば、データ型 char からデータ型 int に修正しようとした場合、そのカラムに制約が含まれていると、alter table modify は次の警告を発行します。

```
Warning: a rule or constraint is defined on column 'new_col' being modified.Verify the
validity of rules and constraints after this ALTER TABLE operation.
```

modify オペレーションは柔軟性に富んでいますが、使用には注意が必要です。一般的に、暗黙的に変換可能なデータ型への修正はエラーを起こすことなく作業できます。明示的に変換可能なデータ型への変換は、テーブル・スキーマでの矛盾を引き起こす可能性があります。カラムのデータ型を修正する前に、sp_depends を使用してカラムレベルの依存性をすべて確認してください。

if exists()...alter table によって生成されるスクリプト

次のような構成体を含むスクリプトは、指定されたカラムが、スクリプトに記述されているテーブルに含まれていない場合にエラーを生成することがあります。

```
if exists (select 1 from syscolumns
     where id = object_id("some_table")
        and name = "some_column")
```

```
begin
    alter table some_table drop some_column
end
```

この例では、バッチが成功するために、some_column が some_table に存在する必要があります。

some_column が some_table にある場合、バッチの最初の実行時に alter table はそのカラムを削除します。その後の実行では、そのバッチはコンパイルされません。

Adaptive Server がこのバッチの前処理中に返すエラーは、通常の select が、存在しないカラムにアクセスしようとしたときに返すエラーと類似しています。このようなエラーは、データ・コピーを必要とする句を使用するテーブルのスキーマを修正するときに返されます。null カラムを追加する場合、前述の構成体を使用すると、Adaptive Server はこのエラーを返しません。

テーブルのスキーマを修正するときにこのようなエラーを発生させないようにするには、次のように、execute immediate コマンドに alter table を含めます。

```
if exists (select 1 from syscolumns
       where id = object_id("some_table")
         and name = "some_column")
begin
    exec ("alter table some_table drop
        some_column")
end
```

execute immediate 文は if exists() 関数が成功した場合のみ実行されるので、Adaptive Server は、このスクリプトをコンパイルするときにエラーを返しません。

たとえば、ロック・スキームの変更などの alter table の他の使用や、コマンドがデータ・コピーを必要としない場合なども、execute immediate 構成体を使用する必要があります。

テーブルおよびその他のオブジェクトの名前の変更

テーブルやその他のデータベース・オブジェクト (カラム、制約、データ型、ビュー、インデックス、ルール、デフォルト、プロシージャ、トリガ) の名前を変更するには、sp_rename を使用します。

ユーザが名前を変更するには、オブジェクトを所有している必要があります。システム・オブジェクトやシステム・データ型の名前を変更することはできません。データベース所有者は、任意のユーザのオブジェクトの名前を変更できます。また、名前を変更するオブジェクトは、現在のデータベースにある必要があります。

データベースの名前を変更するには、sp_renamedb を使用します。『リファレンス・マニュアル：プロシージャ』を参照してください。

たとえば friends_etc の名前を infotable に変更するには、次のように入力します。

```
sp_rename friends_etc, infotable
```

カラムの名前を変更するには、次の構文を使用します。

sp_rename "*table.column*", *newcolumnname*

新しいカラム名にテーブル名のプレフィクスを付けないでください。そうしないと新しい名前が受け入れられません。

インデックスの名前を変更するには、次の構文を使用します。

sp_rename "*table.index*", *newindexname*

新しい名前にはテーブル名を含まないでください。

ユーザ・データ型 tid の名前を t_id に変更するには、次のように入力します。

```
exec sp_rename tid, "t_id"
```

依存するオブジェクトの名前の変更

オブジェクトの名前を変更する場合は、そのオブジェクトに依存するすべてのプロシージャ、トリガ、またはビューのテキストも、新しいオブジェクト名を反映するように変更する必要があります。そのプロシージャ、トリガ、またはビューの名前を変更し、コンパイルするまで、クエリの結果には元のオブジェクト名が表示され続けます。最も安全な方法は、sp_rename を実行するときに「従属」オブジェクトの定義を変更することです。sp_depends を使用すると、従属オブジェクトのリストを使用できます。

defncopy ユーティリティ・プログラムを使用して、プロシージャ、トリガ、ルール、デフォルト、およびビューの定義をオペレーティング・システム・ファイルにコピーできます。このファイルを編集してオブジェクト名を訂正し、defncopy を使用して定義を Adaptive Server にコピーし戻します。『ユーティリティ・ガイド』を参照してください。

テーブルの削除

指定されたテーブルを、その内容およびすべてのインデックスとこれまでに関連付けられた権限とともに、データベースから削除するには、drop table を使用します。テーブルにバインドされたルールとデフォルトはこの時点でバインドされていませんが、他には影響はありません。

テーブルを削除するには、その所有者である必要があります。ただし、テーブルがユーザまたはアプリケーションによって読み書きされている間は、そのテーブルを削除できません。drop table は、master データベースおよびユーザ・データベースの、いずれのシステム・テーブルでも使用できません。

別のデータベースにあるテーブルでも、そのテーブルの所有者であれば削除できます。

テーブル内のすべてのローに対して delete を実行するか、truncate table を使用しても、そのテーブルは、drop を実行するまで存在します。

drop table および truncate table パーミッションは、他のユーザに譲渡することはできません。

計算カラム

計算カラム、計算カラム・インデックス、および関数ベース・インデックスにより、データの操作が容易になり、データへのアクセスが迅速化されます。

- 計算カラムは、同じローの通常カラム、または関数、算術演算子、XML パス・クエリなどを使用した式によって定義されます。

  式は deterministic と nondeterministic のどちらでもかまいません。deterministic 式は、同じ入力のセットから常に同じ結果を返します。

- マテリアライズされた計算カラムには、通常カラムと同じようにインデックスを作成できます。

同様に、計算カラムと関数ベース・インデックスは式でインデックスを作成できます。

計算カラムと関数ベース・インデックスは、いくつかの点で違いがあります。

- 計算カラムは、式に対する省略形とインデックス作成機能の両方を備えています。一方、関数ベース・インデックスは省略形を備えていません。
- 関数ベース・インデックスを使用する場合は、式でインデックスを直接作成できます。一方、計算カラムでインデックスを作成するには、計算カラムを最初に作成する必要があります。
- 計算カラムは deterministic と nondeterministic のどちらでもかまいません。一方、関数ベース・インデックスは deterministic でなければなりません。“deterministic” とは、式の入力値が同じ場合は、戻り値も同じでなければならないということです。「deterministic プロパティ」(323 ページ) を参照してください。
- 計算カラムには、クラスタード・インデックスを作成できます。ただし、関数ベース・インデックスでは作成できません。

関数ベース・インデックスの詳細については、「関数ベース・インデックスを使用したインデックス付け」(416 ページ) を参照してください。

マテリアライズされた計算カラムとマテリアライズされていない計算カラムの違いは、次のとおりです。

- 計算カラムは、マテリアライズすることも、マテリアライズしないこともできます。マテリアライズされたカラムは、ベース・カラムが挿入または更新されたときに事前評価されてテーブルに格納されます。関連する値は、データ・ローとインデックス・ローの両方に格納されます。マテリアライズされたカラムに対するそれ以降のアクセスでは、再評価は必要なく、事前評価された結果がアクセスされます。カラムがマテリアライズされると、そのカラムにアクセスするたびに同じ値が返されます。
- マテリアライズされていないカラムは、仮想カラムと呼ばれることがあります。仮想カラムは、アクセスされた時点でマテリアライズされます。仮想カラムつまりマテリアライズされていないカラムの場合は、カラムがアクセスされるたびに、結果の値を評価する必要があります。つまり、仮想計算カラム式が、nondeterministic　式に基づく式である場合、または nondeterministic の式を呼び出す場合は、アクセスするたびに異なる値が返される可能性があります。また、仮想計算カラムにアクセスすると、ドメイン・エラーのような実行時例外が発生する場合もあります。

計算カラムの使用

計算カラムを使用すると、“Salary + Commission” に対する “Pay” のように、式に対する簡単な表現を作成でき、インデックスを使用できるデータ型の場合は、カラムをインデックス可能にすることができます。インデックスを使用できないデータ型には次のようなものがあります。

- text
- unitext
- image
- Java クラス
- bit

計算カラムは、アプリケーション開発および保守効率の向上を目的としたものです。テーブル定義内の式論理を集中化し、わかりやすいエイリアスを式に与えることによって、計算カラムは非常に簡単で判読可能なクエリを作成します。計算カラムの定義を修正するだけで式を変更できます。

計算カラムは、定義式が nondeterministic 式または関数であるか、nondeterministic 式または関数を呼び出すカラムにインデックスを作成する場合に特に便利です。たとえば、getdate は常に現在の日付を返すため、nondeterministic です。getdate を使用するカラムにインデックスを作成するには、マテリアライズされた計算カラムを構築してからインデックスを作成します。

```
create table rental
      (cust_id int, start_date as getdate()materialized, prod_id int)

create index ind_start_date on rental (start_date)
```

データ型の構成と分解

計算カラムの重要な機能の 1 つは、複雑なデータ型 (たとえば、XML、text、unitext、image、Java クラス) の構成と分解に使用できることです。計算カラムを使用して単純な要素から複雑なデータ型を作成したり (構成)、複雑なデータ型から 1 つ以上の要素を抽出したり (分解) できます。複雑なデータ型は、通常、個別の要素またはフラグメントで構成されています。テーブルを作成する場合、これらの複雑なデータ型を自動的に分解したり構成したりすることを定義できます。たとえば、*order_no*、*part_no*、*customer* のようないくつかのリレーショナル要素とともに、XML "発注" ドキュメントをテーブルに格納するとします。次のように、create table で compute と materialized パラメーターを使用すると、計算カラムでの抽出を定義できます。

```
create table orders(xml_doc image,
order_no compute xml_extract("order_no", xml_doc)materialized,
part_no compute xml_extract ("part_no", xml_doc)materialized,
customer compute xml_extract("customer", xml_doc)materialized)
```

新しい XML ドキュメントをテーブルに挿入するたびに、そのドキュメントのリレーショナル要素が自動的に計算カラムに抽出されます。

また、各ローのリレーショナル・データを XML ドキュメントとして表示するには、テーブル定義で計算カラムを使用し、リレーショナル・データを XML ドキュメントにマッピングするように指定します。たとえば、次のテーブルを定義します。

```
create table orders
(order_no int,part_no int, quantity smallint, customer varchar(50))
```

その後、各ローのリレーショナル・データの XML 表現を返すには、alter table を使用して計算カラムを追加します。

```
alter table orders
add order_xml compute order_xml(order_no, part_no, quantity, customer)
```

次に、select 文を使用して各ローを XML フォーマットで返します。

```
select order_xml from orders
```

ユーザ定義の順序

計算カラムは、複雑なデータ型 (XML、text、unitext、image、Java クラス) の comparison、order by、group by による順序付けをサポートします。計算カラムを使用して複雑なデータのリレーショナル要素を抽出し、それを使用して順序を定義できます。

計算カラムを使用してデータをさまざまなフォーマットに変換し、データ取得時のデータ表現をカスタマイズすることもできます。これは、ユーザ定義のソート順と呼ばれます。たとえば、次のクエリは、サーバのデフォルトの文字セットとソート順の順序 (通常は、ASCII のアルファベット順) で結果を返します。

```
select name, part_no, listPrice from parts_table order by name
```

計算カラムを使用すると、たとえば、ニューヨーク株式市場の銘柄コードのような特殊な略語に基づいた順序など、大文字と小文字を区別しないフォーマットでクエリ結果を表示できます。また、デフォルト以外のシステム・ソート順を使用することもできます。異なる形式にデータを変換するには、組み込み関数 sortkey またはユーザ定義のソート順関数を使用します。

たとえば、ユーザ定義関数 Xform_to_myorder() を使用して *name_in_myorder* という計算カラムを追加するには、次のようにします。

```
alter table parts_table add name_in_myorder compute
     Xform_to_myorder(name)materialized
```

カスタム・フォーマットで結果を返すには：

```
select name, part_no, listPrice from parts_table order by name_in_myorder
```

この方法により、変換済みの順序データをマテリアライズし、インデックスを作成できます。

データ操作言語 (DML: Data Manipulation Language) を使用しても同じ結果が得られます。

```
select name, part_no, listPrice from parts_table
       order by Xform_to_myorder(name)
```

ただし、計算カラムを使用する方法では、変換済みの順序データをマテリアライズしてインデックスを作成できるため、クエリのパフォーマンスが向上します。

意思決定支援システム (DSS)

一般的な意思決定支援システム・アプリケーションでは、集中的なデータ操作、相関、照合データ分析が必要になります。このようなアプリケーションは、クエリで式と関数を頻繁に使用するため、多くの場合特殊なユーザ定義の順序が必要になります。計算カラムおよび関数ベース・インデックスを使用すると、このようなアプリケーションに必要なタスクが簡略化され、パフォーマンスが向上します。

計算カラムの例

計算カラムは式によって定義されます。同じローの通常カラムを結合して式を構築できます。式には、関数、算術演算子、case 式、同じテーブルの他のカラム、グローバル変数、Java オブジェクト、パス式を含めることができます。

下の例では、それぞれ次のようになります。

- part_no は、指定された部品番号を表す Java オブジェクト・カラムです。
- desc は、指定された部品の詳細な説明を含む text カラムです。
- spec は、解析済み XML ストリーム・オブジェクトを格納する image カラムです。
- name_order は、ユーザ定義関数 XML() によって定義される計算カラムです。
- version_order は Java クラスによって定義される計算カラムです。

- descr_index は、des_index() によって定義される計算カラムで、text データのインデックス・キーを生成します。
- spec_index は、xml_index() によって定義される計算カラムで、XML ドキュメントのインデックス・キーを生成します。
- total_cost は、算術式によって定義される計算カラムです。

```
create table parts_table
  (part_no Part.Part_No, name char(30),
  descr text, spec image, listPrice money,
      quantity int,
   name_order compute name_order(part_no)
   version_order compute part_no version,
   descr_index compute des_index(descr),
   spec_index compute xml_index(spec)
   total_cost compute quantity*listPrice
)
```

計算カラムのインデックス

結果のデータ型にインデックスを作成できる場合は、計算カラムのインデックスを作成できます。計算カラム・インデックスおよび関数ベース・インデックスは、XML、text、unitext、image、Java クラスのような複雑なデータ型のインデックスを作成する方法を提供します。

たとえば、以下のコード例は、計算カラムのクラスタード・インデックスを作成します。

```
CREATE CLUSTERED INDEX name_index on part_table(name_order)
CREATE INDEX adt_index on parts_table(version_order)
CREATE INDEX xml_index on parts_table(spec_index)
CREATE INDEX text_index on parts_table(descr_index)
```

インデックスを作成または更新する場合、Adaptive Server は計算カラムを評価し、その結果を使用してインデックスを構築または更新します。

deterministic プロパティ

すべての式および関数は deterministic と nondeterministic のどちらかです。

- 同じ入力値のセットを使用して評価されている場合、deterministic 式および関数は常に同じ結果を返します。次の式は deterministic です。

  ```
  c1 * c2
  ```

- 関数の nondeterministic 式は、同じ入力値のセットを使用して呼び出されている場合でも、評価されるたびに異なる結果を返す可能性があります。関数 getdate は常に現在の日付を返すため、nondeterministic です。

式の deterministic プロパティは、計算カラムまたは関数ベースのインデックス・キーを定義するため、計算カラムまたは関数ベースのインデックス・キー自体を定義します。

deterministic プロパティは、さまざまなシステム関数、ユーザ定義関数、グローバル変数などの nondeterministic 要素が式に含まれるかどうかによって影響されます。

関数が deterministic であるか、nondeterministic であるかは、次のように関数のコーディングによって決まります。

- 関数が nondeterministic 関数を呼び出すと、呼び出した関数自体が nondeterministic 関数になることがある。
- 関数の戻り値が入力値以外の要素によって異なる場合、その関数は nondeterministic である可能性が高い。

deterministic プロパティによる計算カラムへの影響

Adaptive Server には、2 種類の計算カラムがあります。

- 仮想計算カラム
- マテリアライズされた計算カラム

仮想計算カラムはクエリによって参照され、クエリによってアクセスされるたびに評価されます。

マテリアライズされた計算カラムの結果は、データ・ローの挿入時またはベース・カラムの更新時にテーブルに格納されます。マテリアライズされた計算カラムは、クエリ内で参照されても再評価されません。事前評価された結果が使用されます。

- マテリアライズされていない (つまり仮想の) 計算カラムは、インデックス・キーとして使用されると、マテリアライズされた計算カラムになる。
- マテリアライズされた計算カラムは、ベース・カラムのいずれかが更新された場合のみ再評価される。

例

次の例は、nondeterministic 計算カラムとインデックス・キーを使用した場合の有用性と危険性を示しています。すべての例は、説明目的でのみ提供されています。

例 1

この例のテーブル Renting は、さまざまな不動産の賃貸情報を格納します。このテーブルには、次のフィールドが含まれます。

- Cust_ID － 顧客の ID
- Cust_Name － 顧客の名前
- Formatted_Name － 顧客の名前
- Property_ID － 賃貸される不動産の ID
- Property_Name － 標準フォーマットの不動産名
- Start_Date － 賃貸開始日
- Rent_Due － 本日期限の賃貸金額

```
create table Renting
     (Cust_ID int, Cust_Name varchar(30),
     Formatted_Name compute format_name(Cust_Name),
     Property_ID int,Property_Name compute
     get_pname(Property_ID), start_date compute
     today_date()materialized, Rent_due compute
     rent_calculator(Property_ID, Cust_ID,      Start_Date))
```

Formatted_Name、Property_Name、Start_Date、Rent_Due は、計算カラムとして定義されています。

- Formatted_Name － 顧客名を標準フォーマットに変換する仮想計算カラム。出力は入力 Cust_Name のみによって決まるため、Formatted_Name は deterministic です。
- Property_Name － 別のテーブル Property から不動産名を取得する仮想計算カラム。次のように定義されます。

  ```
  create table Property
       (Property_ID int, Property_Name varchar(30),
        Address varchar(50), Rate int)
  ```

  入力された ID に基づいて不動産名を取得するために、関数 get_pname は JDBC クエリを呼び出します。

  ```
  select Property_Name from Property where
  Property_ID=input_ID
  ```

  計算カラム Property_Name は deterministic のように見えますが、実際は nondeterministic です。それは、テーブル Property に格納されたデータおよび入力値 Property_ID によって戻り値が異なるからです。
- Start_Date － 現在の日付を varchar(15) として返す nondeterministic ユーザ定義関数。これは、マテリアライズされたものとして定義されています。したがって、新しいレコードが挿入されるたびに再評価され、再評価された値が Renting テーブルに格納されます。

- Rent_Due − 不動産の賃貸料金、顧客の値引きステータス、賃貸日数に基づいて現在の賃貸料を計算する nondeterministic 仮想計算カラム。

deterministic プロパティによる仮想計算カラムへの影響

Adaptive Server では、定義により、仮想計算カラムが参照されるごとに評価されても、deterministic 仮想計算カラムの繰り返し読み出しが保証されます。たとえばこの文は、テーブルのデータが変更されていない場合は、常に同じ結果を返します。

```
select Cust_ID, Property_ID from Renting
     where Formatted_Name ='RICHARD HUANG'
```

Adaptive Server では、nondeterministic 仮想計算カラムの繰り返し読み出しは保証されません。たとえばこのクエリでは、カラム Rent_Due は別の日の別の結果を返します。このカラムには逐次時間プロパティが設定され、その値は、各賃貸料支払い間の経過時間の関数です。

```
select Cust_Name, Rent_Due from renting
    where Cust_Name= 'RICHARD HUANG'
```

この場合、nondeterministic プロパティは便利ですが、使用に関しては注意が必要です。Start_Date を誤って仮想計算カラムとして定義し、同じクエリを入力すると、すべての不動産を無償で貸し出すことになります。Start_Date は常に現在の日付で評価されるため、このクエリでは Rental_Days の日数は常に 0 になります。

同様に、誤って nondeterministic 計算カラム Rent_Due を事前評価されたカラムとして定義した場合、それをマテリアライズされたものとして宣言するか、インデックス・キーとして使用すると不動産を無償で貸し出すことになります。Rent_Due は、レコードの挿入時に 1 度だけ評価され、賃貸日数は 0 です。この値は、カラムが参照されるたびに返されます。

deterministic プロパティによるマテリアライズされた計算カラムへの影響

マテリアライズされた計算カラムはクエリ内で参照しても再評価されないため、Adaptive Server では、deterministic プロパティに関係なく、マテリアライズされた計算カラムでの繰り返し読み出しが保証されます。代わりに、Adaptive Server では、事前評価された値が使用されます。

どれだけ頻繁に再評価しても、マテリアライズされた deterministic 計算カラムは常に同じ値を持ちます。

マテリアライズされた nondeterministic 計算カラムは、次のルールに従う必要があります。

- 同じ入力のセットを使用している場合でも、同じ計算カラムの評価ごとに異なる結果が返される場合がある。

- 事前評価された nondeterministic 計算カラムへの参照では、現在の評価結果とは異なる可能性がある、事前評価された結果が使用される。つまり、事前評価された nondeterministic 計算カラムでは、現在のデータではなく履歴データが使用される。

例 1 では、Start_Date はマテリアライズされた nondeterministic 計算カラムです。その結果は、ローの挿入日によって異なります。たとえば、賃貸期間が “02/05/04” に始まる場合、“02/05/04” がカラムに挿入され、その後の Start_Date への参照ではこの値が使用されます。その後、06/05/04 にこの値を参照すると、クエリは引き続き “02/05/04” を返します。このカラムに対してクエリを実行するたびに式が評価される場合に期待される “06/05/04” は返されません。

例 2

例 1 で作成したテーブル Renting を使用して、仮想計算カラム Property_Name のインデックスを作成すると、これはマテリアライズされた計算カラムになります。新しいレコードを挿入します。

```
Property_ID=10
```

この新しいレコードは、テーブル Property から get_pname(10) を呼び出し、JDBC クエリが実行されます。

```
select Property_Name from Property where Property_ID=10
```

このクエリは、データ・ローに格納されている “Rose Palace” を返します。他のユーザが次のクエリを発行して不動産名を変更していない場合、このクエリはすべて正常に機能します。

```
update Property set Property_Name ='Red Rose Palace'
     where Property_ID = 10
```

クエリが “Red Rose Palace” を返すため、Adaptive Server は “Red Rose Palace” を格納します。テーブル Property に対するこの update コマンドによって、テーブル Renting 内の Property_Name の格納値が無効になるため、この値も “Red Rose Palace” に更新する必要があります。Property_Name は、テーブル Property のカラム Property_Name ではなく、テーブル Renting のカラム Property_ID によって定義されるため、自動的に更新はされません。その後の Property_Name への参照では、正しくない結果が生成される可能性があります。

この状況を回避するには、テーブル Property にトリガを作成します。

```
CREATE TRIGGER my_t ON Property FOR UPDATE AS
     IF UPDATE(Property_Name)
     BEGIN
     UPDATE Renting SET Renting.Property_ID=Property.Property_ID
          FROM Renting, Property
          WHERE Renting.Property_ID=Property.Property_ID
     END
```

このトリガがテーブル Property のカラム Property_Name を更新すると、Property_Name のベース・カラムである Renting.Property_ID カラムも更新されます。この自動更新によって、Adaptive Server は Property_Name を再評価し、データ・ローに格納された値を更新します。Adaptive Server が テーブル Property のカラム Property_Name を更新するたびに、テーブル Renting のマテリアライズされた計算カラム Property_Name がリフレッシュされて、正しい値が表示されます。

deterministic プロパティによる関数ベース・インデックスへの影響

計算カラムとは異なり、関数ベースのインデックス・キーは deterministic でなければなりません。計算カラムは概念的にはカラムですが、いったん評価および格納されると再評価の必要がありません。しかし、関数や式はクエリ内で使用されるたびに再評価される必要があります。関数が常に同じ入力セットによって同じ結果に評価されないかぎり、インデックス・データなどの事前評価されたデータを使用できません (関数ベースのインデックスの詳細については、「関数ベース・インデックスを使用したインデックス付け」(416 ページ)を参照)。

- Adaptive Server は、内部的には、関数ベースのインデックス・キーをマテリアライズされた隠し計算カラムとして表現します。関数ベースのインデックス・キーの値は、データ・ローとインデックス・ページの両方に格納されるため、マテリアライズされた計算カラムのすべてのプロパティであると想定されます。
- Adaptive Server は、関数ベースまたは式ベースのインデックス・キーは deterministic であると想定します。これらのインデックス・キーがクエリ内で参照されると、インデックス・ページに格納された、事前評価された結果が使用され、インデックス・キーは再評価されません。
- 事前評価された結果は、関数ベースのインデックス・キーのベース・カラムが更新されたときのみ更新されます。
- 例 2 のように、nondeterministic 関数をインデックスとして使用しないでください。予期しない結果になる可能性があります。

ユーザへのパーミッションの割り当て

データベース、テーブル、その他のデータベース・オブジェクトの作成や、特定のコマンドとストアド・プロシージャの実行には、適切なパーミッションが必要です。『リファレンス・マニュアル：コマンド』、『システム管理ガイド 第 1 巻』の「第 17 章 ユーザ・パーミッションの管理」を参照してください。

データベースおよびテーブルの情報を表示する方法

Adaptive Server には、データベース、テーブル、およびその他のデータベース・オブジェクトの情報を表示するためのプロシージャと関数がいくつか用意されています。この項ではそのいくつかについて説明します。ここで説明するプロシージャと関数、およびその他の追加情報については、『リファレンス・マニュアル：プロシージャ』および『リファレンス・マニュアル：ビルディング・ブロック』を参照してください。

データベースに関するヘルプの表示

sp_helpdb は、指定されたデータベースまたはすべての Adaptive Server データベースについての情報をレポートできます。

```
sp_helpdb [dbname]
```

この例では、pubs2 のレポートを表示します。サーバが使用するページ・サイズは 8K です。

```
sp_helpdb pubs2

name      db_size    owner     dbid created        status
--------- ---------- --------- ---- -------------- --------------
pubs2     20.0 MB     sa           4 Apr 25, 2005   select
     into/bulkcopy/pllsort, trunc log on chkpt, mixed log and data

device_fragments    size           usage         created      free kbytes
------------------- -------------- ------------- -----------   ------------
master              10.0MB         data and log  Apr 13 2005          1792
pubs_2_dev         10.0MB         data and log  Apr 13 2005         9888

device                  segment
----------------------- ----------------------
master                  default
master                  logsegment
master                  system
pubs_2_dev              default
pubs_2_dev              logsegment
pubs_2_dev              system
pubs_2_dev              seg1
pubs_2_dev              seg2
```

sp_databases は、サーバ上のすべてのデータベースをリストします。次に例を示します。

```
sp_databases
database_name     database_size  remarks
----------------- -------------  ------------
master                     5120  NULL
model                      2048  NULL
pubs2                      2048  NULL
pubs3                      2048  NULL
```

```
sybsecurity                5120  NULL
sybsystemprocs            30720  NULL
tempdb                     2048  NULL

(7 rows affected, return status = 0)
```

データベースの所有者を表示するには、次のように、sp_helpuser を使用します。

```
sp_helpuser dbo

Users_name    ID_in_db Group_name   Login_name
------------- -------- ------------ ------------
dbo                  1 public       sa
```

現在のデータベースを識別するには、db_id および db_name を使用します。

```
select db_name(), db_id()

------------------------------ ------
master                              1
```

データベース・オブジェクトに関するヘルプの表示

Adaptive Server は、テーブル、カラム、制約などのデータベース・オブジェクトについての情報を返すためのシステム・プロシージャ、カタログ・ストアド・プロシージャ、および組み込み関数を提供します。

データベース・オブジェクトに対する *sp_help* の使用

sp_help を使用すると、指定のデータベース・オブジェクト (sysobjects にリストされたオブジェクト)、指定のデータ型 (systypes にリストされたデータ型)、または現在のデータベースにあるすべてのオブジェクトとデータ型についての情報が表示されます。

sp_help [*objname*]

publishers テーブルについての sp_help 出力を次に示します。

```
Name                Owner         Object_type      Create_date
----------------    -----------   -------------    ------------------------------
publishers          dbo           user table       Nov 9 2004 9:57AM

(1 row affected)
Column_name Type      Length    Prec  Scale   Nulls   Default_name   Rule_name
----------- --------  ------    ----- ------- ------- -------------- ----------
pub_id      char           4    NULL  NULL        0  NULL           pub_idrule
pub_name    varchar       40    NULL  NULL        1  NULL           NULL
city        varchar       20    NULL  NULL        1  NULL           NULL
state       char           2    NULL  NULL        1  NULL           NULL
Access_Rule_name    Computed_Column_object     Identity
------------------- -------------------------  ------------
NULL                NULL                                0
```

```
NULL                NULL                                  0
NULL                NULL                                  0
NULL                NULL                                  0

Object has the following indexes
index_name index_keys  index_description  index_max_rows_per_page
---------- ----------  ---------- ------- -------------
pubind     pub_id      clustered, unique              0

index_fill_factor  index_reservepagegap   index_created      index_local
------------------ ---------------------- ----------------   ----------------
                0                      0  Nov 9 2004 9:58AM  Global Index
(1 row affected)
index_ptn_name   index_ptn_segment
---------------- --------------------
pubind_416001482  default

(1 row affected)
keytype  object      related_object  related_keys
-------  ----------  --------------  ------------
primary  publishers  -- none --      pub_id, *, *, *, *, *
foreign  titles      publishers      pub_id, *, *, *, *, *

(1 row affected)


name         type        partition_type     partitions     partition_keys
--------     --------    ------------------ -------------- ------------------
publishers   base table  roundrobin                     1   NULL

partition_name        partition_id  pages   segment     Create_date
------------------    ------------  ------  ----------- ----------------------
publishers_416001482  416001482         1   default     Nov 9 2004 9:58AM

Partition_Conditions
-----------------------
(NULL)

Avg_pages   Max_pages   Min_pages   Ratio

(return status = 0)


No defined keys for this object.
name    type       partition_type   partitions   partition_keys
------- ---------- ---------------- ------------ ---------------
mytable base table roundrobin                  1 NULL

partiton_name       partition_id   pages     segment   create_date
-----------------   -------------- --------- --------- ------------------------
mytable_1136004047  1136004047             1  default     Nov 29 2004 1:30PM
```

```
partition_conditions
--------------------
(NULL)

Avg_pages   Max_pages   Min_pages   Ratio(Max/Avg)        Ration(Min/Avg)
----------- ----------- ----------- --------------------- -------------------
          1           1           1              1.000000               1.000000
Lock scheme Allpages
The attribute 'exp_row_size' is not applicable to tables with
allpages lock scheme.
The attribute 'concurrency_opt_threshold' is not applicable to
tables with allpages lock scheme.

exp_row_size reservepagegap fillfactor max_rows_per_page identity_gap
 ------------ -------------- ---------- ----------------- ------------
            1              0          0                 0            0
(1 row affected)
concurrency_opt_threshold   optimistic_index_lock   dealloc_first_txtpg
--------------------------- ----------------------- --------------------
                          0                       0                    0
(return status = 0)
```

オブジェクト名を指定しないで sp_help を実行すると、結果のレポートは、sysobjects 内の各オブジェクトの名前、所有者、およびオブジェクト・タイプを表示します。その他に、systypes 内の各ユーザ定義データ型の名前、記憶タイプ、長さ、null 値が許可されているかどうか、およびそのデータ型にバインドされているデフォルトまたはルールも表示されます。また、このレポートは、プライマリ・キーや外部キーのカラムがテーブルやビューに対して定義されているかどうかも示します。

sp_help は、一意性制約または主キー制約を定義することによって作成されたインデックスを含む、テーブル上のインデックスをリストします。ただし、テーブルに定義された整合性制約についての情報は含みません。

sp_helpconstraint によるテーブルの制約情報の表示

sp_helpconstraint は、テーブルに指定されている宣言参照整合性制約についての情報をレポートします。この情報には、制約名と、デフォルト、一意性制約、プライマリ・キー制約、参照制約、または検査制約の定義が含まれます。また、sp_helpconstraint は、指定されたテーブルに関連付けられた参照の数もレポートします。

sp_helpconstraint [*objname*] [, detail]

objname は、問い合わせの対象となるテーブルの名前です。sp_helpconstraint を、テーブル名を含めずに使用すると、現在のデータベースにある各テーブルに関連付けられた参照の数が表示されます。テーブル名を指定すると、sp_helpconstraint は、そのテーブルに関連付けられた名前、定義、および整合性制約の数をレポートします。また、detail オプションは、制約のユーザやエラー・メッセージについての情報を返します。

たとえば、pubs3 の store_employees テーブルの sp_helpconstraint 出力は次のようになります。

```
name                          defn
----------------------------  --------------------------------
store_empl_stor_i_272004000   store_employees FOREIGN KEY
                              (stor_id) REFERENCES stores(stor_id)
store_empl_mgr_id_288004057   store_employees FOREIGN KEY
                              (mgr_id) SELF REFERENCES
                              store_employees(emp_id)
store_empl_2560039432         UNIQUE INDEX( emp_id) :
                              NONCLUSTERED, FOREIGN REFERENCE

(3 rows affected)

Total Number of Referential Constraints: 2
Details:
-- Number of references made by this table: 2
-- Number of references to this table: 1
-- Number of self references to this table: 1

Formula for Calculation:
Total Number of Referential Constraints
= Number of references made by this table
+ Number of references made to this table
- Number of self references within this table
```

現在のデータベース内のテーブルに関連付けられている参照制約の最大数を検出するには、テーブル名を指定しないで sp_helpconstraint を実行します。

```
    sp_helpconstraint

id          name                      Num_referential_constraints
----------- ------------------------- ---------------------------
   80003316 titles                                              4
   16003088 authors                                             3
  176003658 stores                                              3
  256003943 salesdetail                                         3
  208003772 sales                                               2
  336004228 titleauthor                                         2
  896006223 store_employees                                     2
   48003202 publishers                                          1
  128003487 roysched                                            1
  400004456 discounts                                           1
  448004627 au_pix                                              1
  496004798 blurbs                                              1

(11 rows affected)
```

このレポートは、titles テーブルに、pubs3 データベース内で一番多くの参照制約があることを示しています。

テーブルによる領域の使用量の表示

テーブルによる領域の使用量を表示するには、次のように入力します。

```
sp_spaceused [objname]
```

sp_spaceused は、テーブル、クラスタード・インデックス、またはノンクラスタード・インデックスが使用するローおよびデータ・ページの数を計算して表示します。

titles テーブルが使用する領域についてのレポートは、次のように表示されます。

```
sp_spaceused titles

name    rows   reserved  data  index_size  unused
------- -----  --------- ----- ---------   ------
titles  18     48 KB     6 KB  4 KB        38 KB

(0 rows affected)
```

オブジェクト名を指定しない場合、sp_spaceused は、すべてのデータベース・オブジェクトが使用する領域をまとめて表示します。

テーブル、カラム、およびデータ型のリスト

カタログ・ストアド・プロシージャは、システム・テーブルから表形式で情報を取り出します。ユーザは、いくつかのパラメータにワイルドカード文字を指定することができます。

sp_tables を次のフォーマットで使用すると、データベース内のすべてのユーザ・テーブルがリストされます。

```
sp_tables @table_type = "'TABLE'"
```

sp_columns は、データベース内の 1 つ以上のテーブルの任意のカラム、またはすべてのカラムのデータ型を返します。複数のテーブルまたはカラムについての情報を取得するには、ワイルドカード文字を使用できます。

たとえば、次のコマンドは、名前に “sales” が含まれるすべてのテーブルにある、“id” という文字列を含むすべてのカラムについての情報を返します。

```
sp_columns "%sales%", null, null, "%id%"
```

```
table_qualifier table_owner
       table_name     column_name
       data_type type_name  precision  length  scale radix  nullable
      remarks

ss_data_type colid
---------------  -----------
      ----------       -----------
      ---------  --------- ---------  ------  ----- -----  --------
      -------

------------ -----
pubs2              dbo
```

```
       sales            stor_id
       1          char       4           4        NULL  NULL    0
(NULL)

47             1
pubs2             dbo
       salesdetail     stor_id
       1          char         4            4        NULL  NULL    0
(NULL)

4              1
pubs2           dbo
        salesdetail     title_id
        12       varchar       6            6        NULL  NULL 0
    (NULL)

39             3

(3 rows affected, return status = 0)
```

オブジェクト名および ID の表示

オブジェクトの ID と名前を特定するには、object_id() および object_name() を使用します。

```
select object_id("titles")

----------
  208003772
```

オブジェクト名および ID は、sysobjects システム・テーブルに格納されます。

第 9 章

SQL 抽出テーブル

SQL 抽出テーブルは、クエリ式を評価することで 1 つ以上のテーブルから定義され、定義されたクエリ式の中で使用され、クエリの実行中のみ存在します。システム・カタログには記述されず、ディスクに格納されることもありません。

SQL 抽出テーブルを抽象プランの抽出テーブルと混同しないでください。抽象プランの抽出テーブルはクエリの最適化および実行に使用されますが、抽象プランの一部としてのみ存在し、エンド・ユーザには表示されない点で、SQL 抽出テーブルと異なります。

SQL 抽出テーブルは、ネストした select 文から構成される式で作成されます。たとえば、次の例では、抽出テーブル式は pubs2 データベースの publishers テーブルにある都市のリストを返します。

```
select city from (select city from publishers) cities
```

SQL 抽出テーブルは名前が cities であり、city という名前のカラムが 1 つあります。SQL 抽出テーブルは、ネストした select 文によって定義され、次の結果を返すクエリの実行中のみ存続します。

```
city
--------------------
Boston
Washington
Berkeley
```

SQL 抽出テーブルの利点

Colorado で書かれた本のタイトルのみを表示する場合は、次のようなビューを作成します。

```
create view vw_colorado_titles as
     select title
     from titles, titleauthor, authors
     where titles.title_id = titleauthor.title_id
     and titleauthor.au_id = authors.au_id
```

```
      and authors.state ="CO"
```

この結果を表示するために、メモリに格納されているビューである vw_colorado_titles は、次のように繰り返して使用することができます。

```
select * from vw_colorado_titles
```

ビューが不要になれば、次のようにして削除します。

```
drop view vw_colorado_titles
```

クエリ結果が必要なのが 1 回のみの場合には、代わりに次のように SQL 抽出テーブルを使用することもできます。

```
select title
from (select title
      from titles, titleauthor, authors
      where titles.title_id = titleauthor.title_id
      and titleauthor.au_id = authors.au_id and
      authors.state = "CO") dt_colo_titles
```

作成される SQL 抽出テーブルには dt_colo_titles という名前が付けられます。セッション全体を通して存在するテンポラリ・テーブルとは対照的に、SQL 抽出テーブルはクエリの実行中のみ存続します。

1 回のみ必要とされるクエリ結果に関する前述の例では、ビューは SQL 抽出テーブル・クエリよりも望ましくありません。これは、ビューの方が、select 文の他に create 文と drop 文の両方が必要であり、複雑になるためです。1 つのクエリのみに対するビューを作成する利点には、さらに、システム・カタログの取り扱いなどの管理タスクのオーバヘッドに関するものがあります。SQL 抽出テーブルは、管理タスクを必要としない非永続なテーブルを自動的に作成します。複数回使用される SQL 抽出テーブルは、定義がキャッシュされたビューを使用するクエリに匹敵するパフォーマンスを実現します。

SQL 抽出テーブルと最適化

1 つの SQL 文として表されるクエリでは、複数の SQL 文で表されるクエリよりもオプティマイザが有効に活用されます。複数の SQL 文とテンポラリ・テーブルが必要になるようなクエリでも (特に、中間集計結果を保存することが要求されるような場合)、SQL 抽出テーブルは 1 ステップを使用します。たとえば、この例では、1 つの SQL 文が集合演算の結果を SQL 抽出テーブル dt_1 と dt_2 から取得し、この 2 つのテーブル間でジョインを計算します。

```
select dt_1.* from
   (select sum(total_sales)
       from titles_west group by total_sales)
             dt_1(sales_sum),
   (select sum(total_sales)
       from titles_east group by total_sales)
             dt_2(sales_sum)
where dt_1.sales_sum = dt_2.sales_sum
```

SQL 抽出テーブルの構文

SQL 抽出テーブルのクエリ式は、select または select into コマンドの from 句で指定されます。

from_clause ::=
　　from *table_reference* [,*table_reference*]...

table_reference ::=
　　table_view_name | *ANSI_join*

table_view_name ::=
　　{*table_view_reference* | *derived_table_reference*}
　　[holdlock | noholdlock]
　　[readpast]
　　[shared]

table_view_reference ::=
　　[[*database*.]*owner*.]{*table_name* | *view_name*}
　　[[as] *correlation_name*]
　　[index {*index_name* | *table_name* }]
　　[parallel [*degree_of_parallelism*]]
　　[prefetch size]
　　[lru | mru]

derived_table_reference ::=
　　derived_table [as] *correlation_name*
　　['(' *derived_column_list*')']

derived_column_list ::= *column_name* [',' *column_name*] ...

derived_table ::= '(' *select* ')'

抽出テーブル式は、create view 文の select と似ており、次の例外を除いて同じ規則に従います。

- 抽出テーブル式では、create view 文の一部である場合を除き、テンポラリ・テーブルが使用できる。
- 抽出テーブル式では、create view 文の一部である場合を除き、ローカル変数が使用できる。抽出テーブル式内の変数に値を割り当てることはできない。
- 抽出テーブルがカーソルで参照されている場合は、抽出テーブルの構文で変数を create view 文の一部として使用することはできない。
- SQL　抽出テーブルの名前を指定するために抽出テーブル式の後に続ける必要のある correlation_name では抽出カラム・リストを省略できるが、ビューでは名前を指定しないカラムは使用できない。

```
select * from
   (select sum(advance) from total_sales) dt
```

『リファレンス・マニュアル：コマンド』の create view の「使用法」の項の「ビューの制約」を参照してください。

抽出カラム・リスト

抽出カラム・リストが SQL 抽出テーブルに含まれない場合は、SQL 抽出テーブルのカラム名が、抽出テーブル式のターゲット・リストに指定されているカラム名と一致する必要があります。定数式または集計が抽出テーブル式のターゲット・リストに存在する場合のように、カラム名が抽出テーブル式のターゲット・リストで指定されていない場合は、SQL 抽出テーブルに生成されるカラムには名前が付きません。サーバはエラー 11073「抽出テーブル式に null カラム名を含めることはできません...」を返します。

抽出カラム・リストが SQL 抽出テーブルに含まれる場合は、抽出テーブル式のターゲット・リスト内のすべてのカラムの名前を指定してください。これらのカラム名は、クエリ・ブロックで SQL 抽出テーブルの本来のカラム名の代わりに使用する必要があります。カラムは抽出テーブル式に出現した順序でリストする必要があり、カラム名を抽出カラム・リストに複数回指定することはできません。

相関 SQL 抽出テーブル

Transact-SQL は、ANSI 標準ではない相関 SQL 抽出テーブルをサポートしません。たとえば、次のクエリは、dt_publishers1 の抽出テーブル式の内部にある SQL 抽出テーブル dt_publishers2 を参照しているためサポートされません。

```
select * from
   (select * from titles where titles.pub_id =
      dt_publishers2.pub_id) dt_publishers1,
   (select * from publishers where city = "Boston")
      dt_publishers2
where dt_publishers1.pub_id = dt_publishers2.pub_id
```

同様に、次のクエリは、dt_publishers の抽出テーブル式が、SQL 抽出テーブルのスコープ外にある publishers_pub_id カラムを参照しているためサポートされません。

```
select * from publishers
  where pub_id in (select pub_id from
                      (select pub_id from titles
                      where pub_id = publishers.pub_id)

                   dt_publishers)
```

次のクエリは、適切な参照であり、サポートされています。

```
select * from publishers
  where pub_id in (select pub_id from
                      (select pub_id from titles)
                   dt_publishers
                     where pub_id = publishers.pub_id)
```

SQL 抽出テーブルの使用

SQL 抽出テーブルを使用して、多様な SQL 句と演算子を使用している統合された大きなクエリの一部を形成できます。

ネスト

クエリでは、ネストした多数の抽出テーブル式である、SQL 抽出テーブルを定義する SQL 式を使用できます。次の例では、最も内側の抽出テーブル式は SQL 抽出テーブル dt_1 を定義しており、この select from で SQL 抽出テーブル dt_2 を定義する抽出テーブル式が形成されます。

```
select postalcode
   from (select postalcode
            from (select postalcode
                    from authors) dt_1) dt_2
```

ネストの段階は 25 までに制限されています。

SQL 抽出テーブルを使用したサブクエリ

SQL 抽出テーブルは、サブクエリの from 句で使用できます。たとえば、次のクエリはビジネス書を出版した出版社の名前を検索します。

```
select pub_name from publishers
   where "business" in
      (select type from
         (select type from titles, publishers
             where titles.pub_id = publishers.pub_id)
       dt_titles)
```

ここで、dt_titles は最も内側の select 文で定義された SQL 抽出テーブルです。

SQL 抽出テーブルは、サブクエリの使用が有効な場所であればどこでも、サブクエリの from 句で使用できます。「第 5 章 サブクエリ：他のクエリ内でのクエリの使用」を参照してください。

union

union 句は抽出テーブル式内で使用できます。たとえば、次のクエリは、sales テーブルと sales_east テーブルの両方の stor_id カラムと ord_num カラムの内容を生成します。

```
select * from
   (select stor_id, ord_num from sales
    union
    select stor_id, ord_num from sales_east)
dt_sales_info
```

ここでは、2 つの select 演算の union で SQL 抽出テーブル dt_sales_info を定義します。

サブクエリ内の union

union 句は、抽出テーブル式の内部のサブクエリで使用できます。次の例では、SQL 抽出テーブル内のサブクエリで union 句を使用して、sales テーブルと sales_east テーブルにリストされている各書店で売れた本のタイトルをリストします。

```
select title_id from salesdetail
   where stor_id in
      (select stor_id from
         (select stor_id from sales
          union
          select stor_id from sales_east)
       dt_stores)
```

SQL 抽出テーブルでのカラム名の変更

抽出カラム・リストが SQL 抽出テーブルに対して含まれる場合は、次の例のように SQL 抽出テーブルの名前の後に続け、カッコで囲みます。

```
select dt_b.book_title, dt_b.tot_sales
   from (select title, total_sales
            from titles) dt_b (book_title, tot_sales)
   where dt_b.book_title like "%Computer%"
```

ここでは、抽出テーブル式内のカラム名 title と total_sales は、抽出カラム・リストを使用してそれぞれ book_title と tot_sales に変更されます。book_title カラム名と tot_sales カラム名はクエリの他の部分で使用されます。

注意 SQL 抽出テーブルには、名前のないカラムを作成できません。

定数式

カラム名に定数式が使用される場合のように、抽出テーブル式のターゲット・リストでカラム名が指定されていない場合は、SQL 抽出テーブルに生成されるカラムには名前が付きません。

```
1> select * from
2> (select title_id, (lorange + hirange)/2
3> from roysched) as dt_avg_range
4> go
title_id
--------- -----------
```

```
BU1032    2500
BU1032    27500
PC1035    1000
PC1035    2500
```

抽出カラム・リストを使用すると、抽出テーブル式のターゲット・リストのカラム名を指定できます。

```
1> select * from
2> (select title_id, (lorange + hirange)/2
3> from roysched) as dt_avg_range (title, avg_range)
4> go
title     avg_range
--------- -----------
BU1032    2500
BU1032    27500
PC1035    1000
PC1035    2500
```

または、抽出テーブル式のターゲット・リストのカラム名を変更することにより、カラム名を指定できます。

```
1> select * from
2> (select title_id, (lorange + hirange)/2 avg_range
3> from roysched) as dt_avg_range
4> go
title     avg_range
--------- -----------
BU1032    2500
BU1032    27500
PC1035    1000
PC1035    2500
```

注意 抽出カラム・リストと抽出テーブル式のターゲット・リストの両方でカラム名を指定した場合、生成されるカラムの名前は抽出カラム・リストによって付けられます。抽出カラム・リストのカラム名は、抽出テーブル式のターゲット・リストに指定された名前に優先します。

---

create view 文内で定数式を使用した場合は、定数式の結果に対してカラム名を指定します。

集合関数

抽出テーブル式では、sum、avg、max、min、count_big、count などの集合関数を使用できます。次の例は、SQL 抽出テーブル dt_a からカラム pub_id と adv_sum を選択します。2 番目のカラムは、titles テーブルの advance カラムに対して sum 関数を使用して抽出テーブル式に作成されます。

```
select dt_a.pub_id, dt_a.adv_sum
   from (select pub_id, sum(advance) adv_sum
         from titles group by pub_id) dt_a
```

create view 文内で集合関数を使用した場合は、集約結果に対してカラム名を指定します。

SQL 抽出テーブルとのジョイン

次の例は、SQL 抽出テーブルと既存のテーブルの間のジョインを示します。ジョインは where 句で指定されます。ジョインされる 2 つのテーブルは dt_c (SQL 抽出テーブル) と publishers (pubs2 データベース内の既存のテーブル) です。

```
select dt_c.title_id, dt_c.pub_id
   from (select title_id, pub_id from titles) as dt_c,
        publishers
   where dt_c.pub_id = publishers.pub_id
```

次の例は、2 つの SQL 抽出テーブル間のジョインを示します。ジョインされる 2 つのテーブルは、dt_c と dt_d です。

```
select dt_c.title_id, dt_c.pub_id
   from (select title_id, pub_id from titles)
         as dt_c,
        (select pub_id from publishers)
         as dt_d
   where dt_c.pub_id = dt_d.pub_id
```

SQL 抽出テーブルを伴う外部ジョインも可能です。Sybase は、左右両方の外部ジョインをサポートします。次の例は、2 つの SQL 抽出テーブル間の左外部ジョインを示します。

```
select dt_c.title_id, dt_c.pub_id
   from (select title_id, pub_id from titles)
         as dt_c,
        (select title_id, pub_id from publishers)
         as dt_d
   where dt_c.title_id *= dt_d.title_id
```

次の例は、抽出テーブル式内の左外部ジョインを示します。

```
select dt_titles.title_id
   from (select * from titles, titleauthor
         where titles.title_id *= titleauthor.title_id)
dt_titles
```

SQL 抽出テーブルからのテーブルの作成

SQL 抽出テーブルから取得したデータが新しいテーブルに挿入されます。

```
select pubdate into pub_dates
   from (select pubdate from titles) dt_e
            where pubdate = "450128 12:30:1PM"
```

SQL 抽出テーブル dt_e からのデータが新しいテーブルである pub_dates に挿入されます。

SQL 抽出テーブルでのビューの使用

次の例は、SQL 抽出テーブル dt_colo_pubs を使用してビュー view_colo_publishers を作成し、Colorado を拠点とする出版社を表示します。

```
create view view_colo_publishers (Pub_Id, Publisher,
City, State)
as select pub_id, pub_name, city, state
from
(select * from publishers where state="CO")
dt_colo_pubs
```

抽出テーブル式の insert ルールとパーミッション設定が create view 文の select 部分の insert ルールとパーミッション設定に従っている場合は、SQL 抽出テーブルを含むビューからデータを挿入できます。たとえば、次の insert 文は view_colo_publishers ビューを通してビューのベースとなっている publishers テーブルにローを挿入します。

```
insert view_colo_publishers
values ('1799', 'Gigantico Publications', 'Denver',
'CO')
```

SQL 抽出テーブルを使用するビューを通して既存のデータを更新することもできます。

```
update view_colo_publishers
set Publisher = "Colossicorp Industries"
where Pub_Id = "1699"
```

注意 カラム名は、基本となるテーブルのカラム名ではなく、ビュー定義のカラム名を指定してください。

SQL 抽出テーブルを使用するビューは、標準の方法で削除されます。

```
drop view view_colo_publishers
```

相関属性

SQL 抽出テーブルのスコープ外の相関属性は、SQL 抽出テーブル式からは参照できません。たとえば、次のクエリではエラー・メッセージが表示されます。

```
select * from publishers
   where pub_id in
      (select pub_id from
         (select pub_id from titles
          where pub_id = publishers.pub_id)
       dt_publishers)
```

ここでは、カラム publishers.pub_id は SQL 抽出テーブル式で参照されますが、これは SQL 抽出テーブル dt_publishers のスコープ外です。

第 10 章

テーブルとインデックスの分割

パーティションを使用して、テーブルとインデックスを小さく分割して管理しやすい大きさにして、大きいテーブルとインデックスを管理します。パーティションを使用すると、大規模なインデックスを使用する場合と同様に、データへのアクセスが高速化され、かつアクセスしやすくなります。

各パーティションは独立したセグメントに配置できます。パーティションはデータベース・オブジェクトであり、独立して管理できます。たとえば、パーティション・レベルでデータをロードし、インデックスを作成できます。また、パーティションはエンド・ユーザに対しては透過的であり、テーブルがパーティションに分割されているかどうかに関係なく、同じ DML コマンドを使用してデータの選択、挿入、削除を行うことができます。

Adaptive Server は水平分割をサポートしています。この機能により、選択したテーブル・ローをディスク・デバイス間に分散できます。個々のテーブルやインデックス・ローは、パーティション方式に基づいて、パーティションに割り当てられます。

分割は並列処理の基盤であり、パフォーマンスの大幅な向上をもたらします。

注意 セマンティックベースの分割機能は、別途ライセンスされます。ライセンスのあるサイトでセマンティック分割を有効にするには、enable semantic partitioning 設定パラメータの値を 1 に設定します。『システム管理ガイド 第 1 巻』の「第 5 章 設定パラメータ」を参照してください。

分割機能には、次のような利点があります。

- スケーラビリティの向上
- パフォーマンスの向上 － 異なるパーティションでの複数の同時 I/O、複数のパーティションで同時に動作する複数の CPU で複数のスレッドを使用可能
- 応答時間の短縮
- アプリケーションに対するパーティションの透過性
- 大容量データベース (VLDB: Very Large Database) のサポート － 大規模なテーブルの複数のパーティションの同時スキャン
- 範囲分割による履歴データの管理

注意 デフォルトでは、単一のパーティションを持つテーブルが作成され、ラウンドロビン分割方式が使用されます。分割構文を使用せずに作成または修正されたテーブル (デフォルト) と分割構文を使用して作成されたテーブルを区別するため、これらのテーブルを「分割されていない」テーブルと呼びます。

12.5.x 以前の Adaptive Server からのアップグレード

バージョン 12.5.x 以前から Adaptive Server 15.0 以降にアップグレードすると、すべてのデータベース・テーブルが、分割されていないテーブルに変更されます。インデックスは変更されず、分割されていないグローバル・インデックスとして保持されます。

データベース・テーブルを再分割したり、インデックスを分割したりする場合は、この章の手順に従ってください。

データ・パーティション

データ・パーティションは、ユニークなパーティション ID を持つ独立したデータベース・オブジェクトです。データ・パーティションは、テーブルのサブセットであり、カラム定義と参照整合性の制約はベース・テーブルと共通です。I/O の並列処理を最大化するため、各パーティションを異なるセグメントにバインドし、各セグメントを異なる記憶デバイスにバインドすることをおすすめします。

分割キー

各セマンティック分割テーブルには、個々のデータ・ローを異なるパーティションに分散する方法を決定する分割キーがあります。分割キーは、単一の分割キー・カラムまたは複数のキー・カラムで構成することができます。キー・カラムの値によって、実際のパーティションの分散方法が決まります。

範囲およびハッシュにより分割されたテーブルは、分割キー内に最大で 31 個のキー・カラムを持つことができます。リスト・パーティションは、分割キー内にキー・カラムを 1 つ持つことができます。ラウンドロビン分割テーブルは分割キーを持ちません。

以下の型の分割キー・カラムは指定できません。

- text、image、unitext
- bit
- Java クラス
- 計算カラム

これらのデータ型のカラムを含むテーブルは分割できます。ただし、分割キー・カラムは、サポートされているデータ型である必要があります。

インデックス・パーティション

テーブルと同様、インデックスも分割できます。15.0 よりも前のバージョンの Adaptive Server では、インデックスはすべてグローバルでした。Adaptive Server 15.0 以降では、グローバル・インデックスだけでなく、ローカル・インデックスも作成できます。

インデックス・パーティションは、インデックス ID とパーティション ID のユニークな組み合わせで識別される独立したデータベース・オブジェクトです。インデックスのサブセットであり、セグメントまたは他の記憶デバイスに配置されます。

Adaptive Server では、ローカル・インデックスとグローバル・インデックスがサポートされています。

- 「ローカル・インデックス」– 単一のデータ・パーティションを対象とします。セマンティック分割テーブルの場合、ローカル・インデックスは、ベース・テーブルに対して等分割されたパーティションを持ちます。つまり、テーブルとインデックスは同じ分割キーと分割方式を共有します。

  分割されたテーブルにローカル・インデックスが作成されている場合、各ローカル・インデックス・パーティションは、対応するデータ・パーティションを 1 つだけ持ちます。

- 「グローバル・インデックス」– テーブルのすべてのデータ・パーティションを対象とします。分割されていないグローバル・インデックスだけがサポートされています。分割されていないテーブルの分割されていないインデックスは、すべてグローバル・インデックスです。

分割されたテーブルでは、分割されたインデックスと分割されていないインデックスを次のように混在させることができます。

- 分割されたテーブルは、分割されたインデックスと分割されていないインデックスを持つことができる。
- 分割されていないテーブルは、分割されていないグローバル・インデックスのみを持つことができる。

パーティション ID

パーティション ID は、オブジェクト ID に似た疑似ランダム番号です。パーティション ID とオブジェクト ID は、同じ番号空間から割り当てられます。インデックス・パーティションやデータ・パーティションは、インデックス ID とパーティション ID のユニークな組み合わせで識別されます。

ロックとパーティション

共有モードまたは排他モードで DDL コマンドを実行すると、操作が特定のパーティションに影響するかどうかにかかわらず、Adaptive Server はテーブル全体をロックします。Adaptive Server は、個々のパーティションをロックしません。

分割方式

Adaptive Server は、4 つのデータ分割方式をサポートしています。

- 範囲分割
- ハッシュ分割
- リスト分割
- ラウンドロビン分割

範囲分割

範囲分割されたテーブルやインデックスのローは、分割キー・カラムの値に基づいて各パーティションに分散されます。各ローの分割カラム値は、上限値と下限値の組み合わせと比較され、ローが属するパーティションが決定されます。

- 各パーティションには、パーティションの作成時に values <= 句によって指定された上限値がある。
- 最初に作成したパーティション以外の各パーティションには、その下のパーティションの values <= 句によって暗黙的に指定された下限値がある。

範囲分割は、OLTP や意思決定支援環境における高パフォーマンス・アプリケーションで特に役立ちます。すべてのパーティションに均等にローが割り当てられるように、慎重に範囲を選択してください。パーティション間の負荷を均等に分散するには、分割キー・カラムのデータ分散に関する知識が重要です。

範囲分割したパーティションには順序があります。つまり、後続の各パーティションは直前のパーティションよりも大きい境界値を持つ必要があります。

ハッシュ分割

ハッシュ分割では、Adaptive Server は、ハッシュ関数を使用して各ローのパーティションの割り当てを指定します。分割キー・カラムはユーザが選択しますが、パーティションの割り当てを制御するハッシュ関数は Adaptive Server によって選択されます。

ハッシュ分割は、以下の場合に適切です。

- 多数のパーティションを持つ大規模なテーブル (特に、意思決定支援環境内)
- ハッシュ・キー・カラムでの効率的な等価探索
- 特に順序付けのないデータ (たとえば、英数字の製品コード・キー)

適切な分割キーを選択すると、ハッシュ分割によってすべてのパーティションに均等にデータが分散されます。ただし、不適切なキー (たとえば、多数のローに対して同じ値を持つキー) を選択すると、パーティション間のローの分散のバランスがとれないため、データが偏る可能性があります。

リスト分割

範囲分割と同様、リスト分割では、ローはセマンティック的に (つまり、分割キー・カラムの実際の値に基づいて) 分散されます。リスト・パーティションでは、キー・カラムを 1 つしか使用できません。分割キー・カラムの値は、ユーザが指定した値のセットと比較され、各ローが属するパーティションが決定されます。分割キーは、パーティションに指定された値のいずれかと正確に一致している必要があります。

各パーティションの値リストは値を 1 つ以上含みます。値リストは、すべてのパーティション間でユニークである必要があります。各リスト・パーティションには、最大で 250 個の値を指定できます。リスト・パーティションは順序付けられていません。

ラウンドロビン分割

ラウンドロビン分割では、各パーティションに同じくらいの数のローが含まれ、負荷分散が実現されるように、ラウンドロビン方式で各パーティションにローが割り当てられます。分割キーがないため、ローはすべてのパーティションにランダムに分配されます。

ラウンドロビン分割

- 将来の挿入のための複数の挿入ポイント
- 並列処理を使用してパフォーマンスを強化する方法
- 個々のパーティションの統計情報の更新やデータのトランケートなど、管理タスクを実行する方法

複合分割キー

セマンティック分割テーブルは、テーブルごとまたはインデックスごとに 1 つの分割キーを持ちます。範囲分割またはハッシュ分割されたテーブルの場合、分割キーは最大で 31 個のキー・カラムを持つ複合キーです。ハッシュ分割されたテーブルが複合分割キーを持つ場合、Adaptive Server は、すべての分割キー・カラムの値を取得し、システムから提供されるハッシュ関数を使用して、結果として生成されるデータ・ストリームをハッシュします。

範囲分割されたテーブルが複数の分割キー・カラムを持つ場合、Adaptive Server は、各データ・ローの分割キー・カラムに対応する値を各パーティションの上限値および下限値と比較します。各パーティションの境界は、分割キー・カラムごとに 1 つの値を含む、1 つ以上の値のリストです。

Adaptive Server は、テーブルを最初に作成したときに指定された順序で、分割キーの値を境界値と比較します。最初のキー値がパーティションの割り当て基準を満たしている場合、ローはそのパーティションに割り当てられ、その他のキー値は評価されません。最初のキー値が割り当て基準を満たしていない場合、割り当て基準が満たされるまで後続のキー値が評価されます。このようにして、最も少ない場合は 1 つの分割キー、最も多い場合はすべてのキー値が評価され、パーティションの割り当てが決まります。

たとえば、key1 と key2 が、my_table のカラムを分割しているとします。テーブルは、p1、p2、p3 の 3 つのパーティションで構成されます。p1 には (a, b)、p2 には (c, d)、p3 には (e, f) が上限値として宣言されています。

```
if key1 < a, then the row is assigned to p1
if key1 = a, then
    if key2 <b or key2 = b, then the row is assigned to p1
key1 > a or (key1 = a and key2 > b), then
    if key1 < c, then the row is assigned to p2
        if key1 = c, then
            if key2 < d or key2 = d, then the row is assigned to p2
        if key1 > c or (key1 = c and key2 > d), then
            if key1 < e, then the row is assigned to p3
            if key1 = e, then
                if key2 < f or key2 = f, then the row is assigned to p3
                if key2 > f, then the row is not assigned
```

pubs2 の roysched テーブルは範囲分割されているとします。分割カラムは、「高範囲 (hirange)」と「印税 (royalty)」です。パーティションとしては、p1、p2、p3 の 3 つがあります。上限値は、p1 が (5000, 14)、p2 が (10000, 10)、p3 が (100000, 25) です。

alter table を使用して、roysched テーブル内にパーティションを作成できます。

```
alter table roysched partition
        by range (hirange, royalty)
    (p1 values <= (5000, 14),
    p2 values <= (10000, 10),
    p3 values <= (100000, 25))
```

ローは次のように分割されます。

- 分割キー値が (4001, 12)、(5000, 12)、(5000, 14)、(3000, 18) のローは p1 に割り当てられる。(5000, 12), (5000, 14), (3000, 18).
- 分割キー値が (6000, 18)、(5000, 15)、(5500, 22)、(10000, 10)、(10000, 9) のローは p2 に割り当てられる。(5000, 15), (5500, 22), (10000, 10), (10000, 9).
- 分割キー値が (10000, 22)、(80000, 24)、(100000, 2)、(100000, 16) のローは p3 に割り当てられる。

分割キー・カラムを 3 つ以上持つテーブルも同じように評価されます。

パーティション排除

セマンティックベースの分割を行うと、Adaptive Server は検索の実行時に特定のパーティションを削除できます。たとえば、範囲ベースのパーティションには、分割キーが個別の値セットであるローが含まれます。クエリの述部 (where 句) がこのような分割キーに基づいている場合、Adaptive Server は、特定のパーティションがクエリを満たす可能性があるかどうかを直ちに確認します。この動作は「パーティション排除」または「パーティション除去」と呼ばれ、実行時に時間とリソースを大幅に節約できます。

- 範囲およびリスト分割の場合 － 単一のテーブルの分割キー・カラムにある等号述語 (=) および範囲述語 (>、>=、<、<=) にパーティション排除を適用できます。
- ハッシュ分割の場合 － 単一のテーブルの等号述語にのみパーティション排除を適用できます。
- 範囲、リスト、およびハッシュ分割の場合 － 「等しくない (!=)」句を持つ述語や、パーティション・カラムに式を持つ複雑な述語にはパーティション排除を適用できません。

たとえば、pubs2 の roysched テーブルが hirange と royalty で分割されているとします (「複合分割キー」(352 ページ) 参照)。Adaptive Server は、パーティション排除を次のクエリで使用できます。

```
select avg(royalty) from roysched
where hirange <= 10000 and royalty < 9
```

パーティション排除プロセスは、このクエリの条件を満たすパーティションとして p1 と p2 だけを識別します。つまり、p3 パーティションはスキャンする必要がありません。したがって、Adaptive Server は p1 と p2 だけをスキャンすればよいため、より効率的にクエリ結果を返すことができます。

次の例では、Adaptive Server はパーティション排除を使用しません。

```
select * from roysched
where hirange != 5000
select * from roysched
where royalty*0.15 >= 45
```

注意 逐次実行モードでは、パーティション排除は、スキャン、挿入、削除、および更新に対してのみ適用されます。その他の演算子には適用されません。並列実行モードでは、パーティション排除はすべての演算子に適用されます。

インデックスとパーティション

Adaptive Server がデータを見つけるときにインデックスは役立ちます。インデックスはディスク上にあるテーブル・カラムのデータの位置を指すため、データの検索が速くなります。グローバル・インデックスとローカル・インデックスを作成できます。これらは、それぞれクラスタード・インデックスまたはノンクラスタード・インデックスとして使用できます。

クラスタード・インデックスでは、物理データはインデックスと同じ順序で格納され、インデックスの最下位レベルには実際のデータ・ページが存在します。ノンクラスタード・インデックスでは、物理データはインデックスと同じ順序で格納されず、インデックスの最下位レベルにはデータ・ページのローへのポインタが存在します。

セマンティック分割テーブルのクラスタード・インデックスは、常にローカル・インデックスです。create index コマンドで “local” インデックスが指定されているかどうかは関係ありません。ラウンドロビン・テーブルのクラスタード・インデックスは、グローバル・インデックス、ローカル・インデックスのいずれかです。

グローバル・インデックス

グローバル・インデックスは、ベース・テーブルに対して等分割されていない 1 つ以上のパーティションのデータを対象とします。分割されていないグローバル・インデックスだけがサポートされます。グローバル・インデックスはすべてのパーティションを対象とします。

15.0 よりも前のバージョンの Adaptive Server では、分割されたテーブルに対して作成されたインデックスはすべてグローバル・インデックスでした。グローバル・インデックスは、以前のバージョンの Adaptive Server との互換性を保つためにサポートされていますが、OLTP 環境で特に役立ちます。

Adaptive Server は、次の種類のグローバル・インデックスをサポートします。

- ラウンドロビン・テーブルおよび分割されていないテーブルのクラスタード・インデックス
- すべての種類のテーブルのノンクラスタード・インデックス

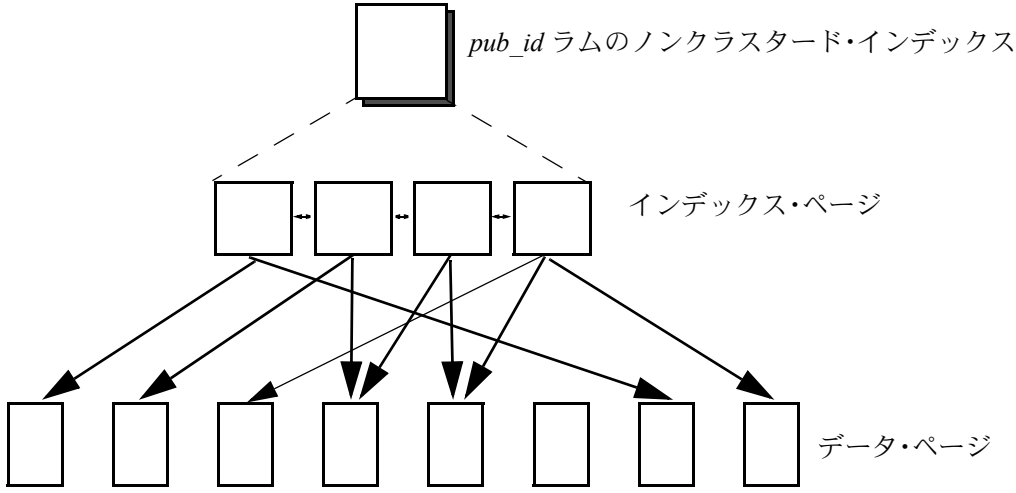
分割されていないテーブルのグローバル・ノンクラスタード・インデックス

図 10-1 の例は、Adaptive Server 12.5 以降でサポートされているデフォルトのノンクラスタード・インデックス設定を示しています。

分割されていない publishers テーブルに対してこのインデックスを作成するには、次のように入力します。

```
create nonclustered index publish5_idx on
    publishers(pub_id)
```

図 10-1: 分割されていないテーブルのグローバル・ノンクラスタード・インデックス



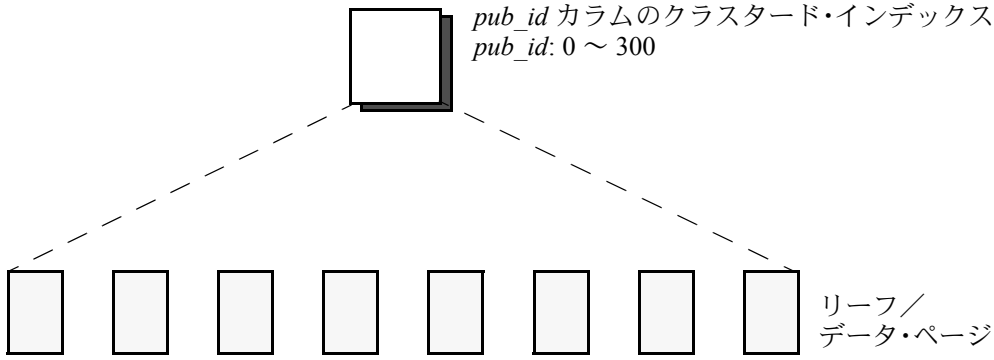
分割されていないテーブルのグローバル・クラスタード・インデックス

図 10-2 は、デフォルトのクラスタード・インデックス設定を示しています。テーブルおよびインデックスは分割されていません。

分割されていない publishers テーブルに対してこのテーブルを作成するには、次のように入力します。

```
create clustered index publish4_idx
    on publishers(pub_id)
```

図 10-2: 分割されていないテーブルのグローバル・クラスタード・インデックス



ラウンドロビン方式で分割されたテーブルのグローバル・クラスタード・インデックス

Adaptive Server では、ラウンドロビン方式で分割されたテーブルに対してのみ、分割されていないクラスタード・グローバル・インデックスがサポートされています。

注意 ラウンドロビン・テーブルのパーティション数が 255 を超える場合、そのテーブルにはグローバル・インデックスを作成できません。ローカル・インデックスを作成する必要があります。

分割されていないインデックスでは、すべてのパーティションに対して完全なテーブル・スキャンを実行できます。リーフ・インデックス・ページは、全ページロック・テーブルのデータ・ページでもあるため、このインデックスは、すべてのデータ・パーティションが同じセグメントに存在する場合に最も役立ちます。データ分割キーのインデックスを作成する必要があります。

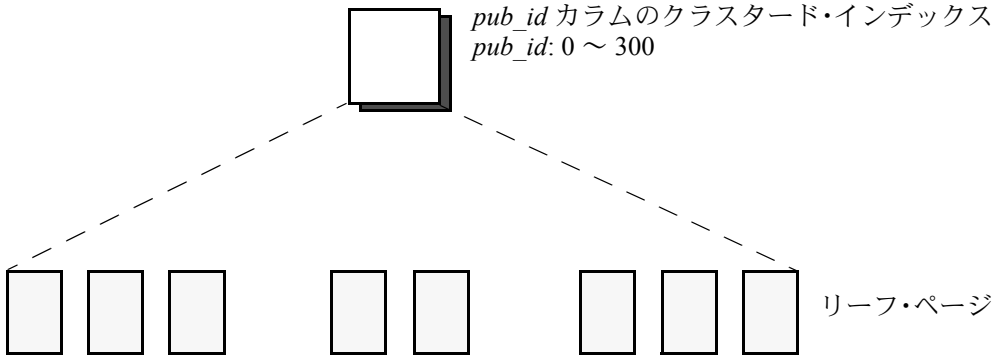
この例では (図 10-3)、pubs2 の publishers テーブルに 3 つのラウンドロビン・パーティションを作成します。パーティションを作成する前に、すべてのインデックスを削除します。

```
alter table publishers partition 3
```

ラウンド・ロビン方式で分割された publishers テーブルにクラスタード・インデックスを作成するには、次のように入力します。

```
create clustered index publish1_idx
    on publishers(pub_id)
```

図 10-3: ラウンドロビン方式で分割されたテーブルのグローバル・クラスタード・インデックス



分割されたテーブルのグローバル・ノンクラスタード・インデックス

すべてのテーブル分割方式について、分割されていないノンクラスタード・グローバル・インデックスを作成できます。

インデックス・パーティションとデータ・パーティションは、同じセグメントまたは異なるセグメントに配置できます。テーブル内のインデックスを使用できるすべてのカラムにインデックスを作成できます。

図 10-4 の例では、pub_name カラムに対してインデックスが作成されています。テーブルは pub_id カラムで分割されています。

この例では、alter table を使用して、pub_id カラムの 3 つの範囲パーティションで publishers を再分割します。

```
alter table publishers partition by range(pub_id)
(a values <= ("100"),
b values <= ("200"),
c values <= ("300"))
```

pub_name カラムのグローバル・ノンクラスタード・インデックスを作成するには、次のように入力します。

```
create nonclustered index publish2_idx
    on publishers(pub_name)
```

図 10-4: 分割されたテーブルのグローバル・ノンクラスタード・インデックス

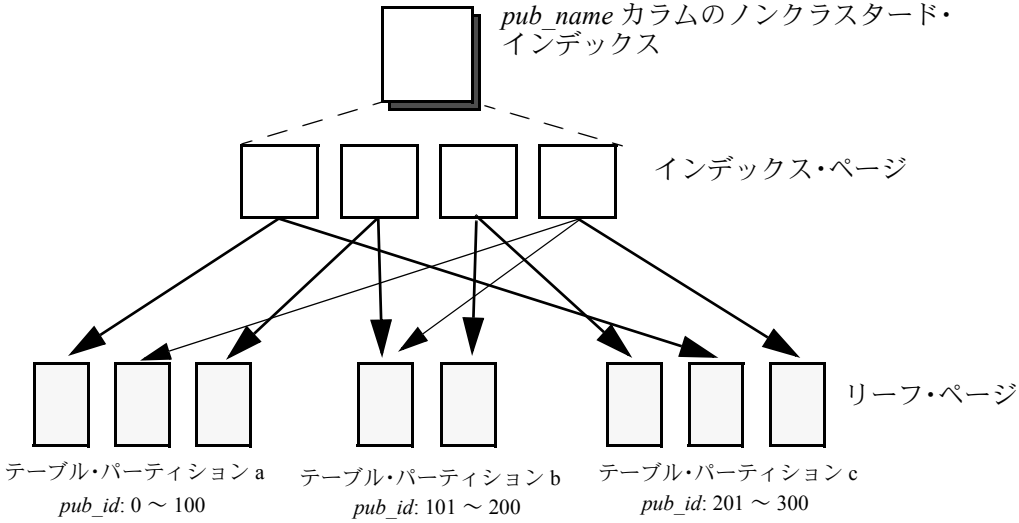
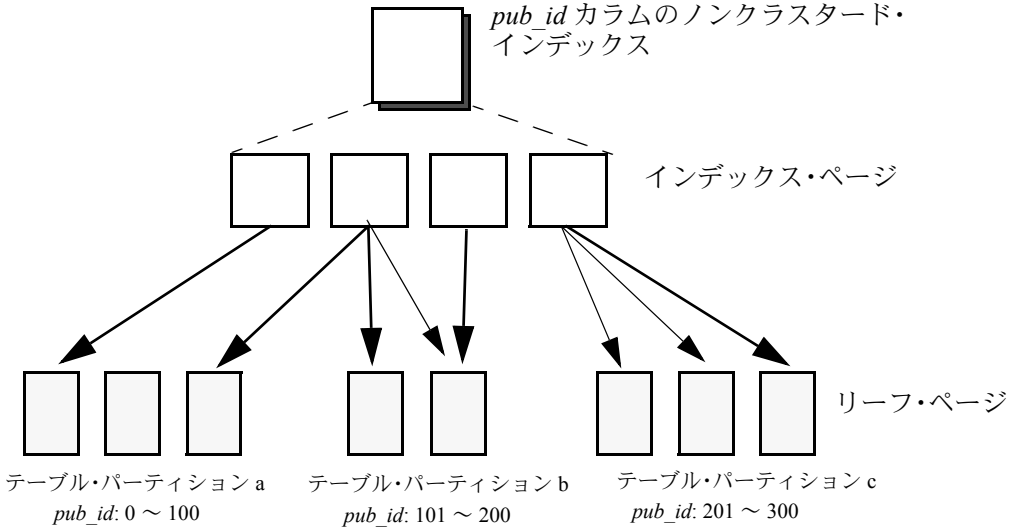


図 10-5 の例では、pub_id カラムのインデックスが作成されています。テーブルも pub_id カラムで分割されています。

pub_id カラムのグローバル・ノンクラスタード・インデックスを作成するには、次のように入力します。

```
create nonclustered index publish3_idx
    on publishers(pub_id)
```

図 10-5: 分割されたテーブルのグローバル・ノンクラスタード・インデックス



ローカル・インデックス

すべてのローカル・インデックスは、ベース・テーブルのデータ・パーティションに対して等分割されます。つまり、ベース・テーブルの分割方式と分割キーを継承します。各ローカル・インデックス・パーティションの範囲は、単一のデータ・パーティションのみです。ローカル・インデックスは、範囲、ハッシュ、リスト、ラウンドロビン方式で分割されたテーブルに対して作成できます。ローカル・インデックスを使用すると、複数のスレッドが各データ・パーティションを並列でスキャンできるため、パフォーマンスが大幅に向上する可能性があります。

ローカル・クラスタード・インデックス

テーブルを分割すると、値に基づいてローがパーティションに割り当てられますが、データはソートされません。ローカル・インデックスを作成すると、各パーティションは個別にソートされます。

各データ・パーティションに関する情報は、パーティションの作成時に設定された境界値に準拠します。これにより、テーブル全体にわたってユニークなインデックス・キーを強制できます。

図 10-6 は、分割されたテーブルの、分割されたクラスタード・インデックスの例を示しています。インデックスは、pub_id カラム上で作成され、テーブルは pub_id 上で分インデックス化されます。この例では、pub_id カラムでユニーク性を強制することができます。

範囲分割された publishers テーブル上にこのテーブルを作成するには、次のように入力します。

```
create clustered index publish6_idx
    on publishers(pub_id)
    local index p1, p2, p3
```

図 10-6: ユニークなローカル・クラスタード・インデックス

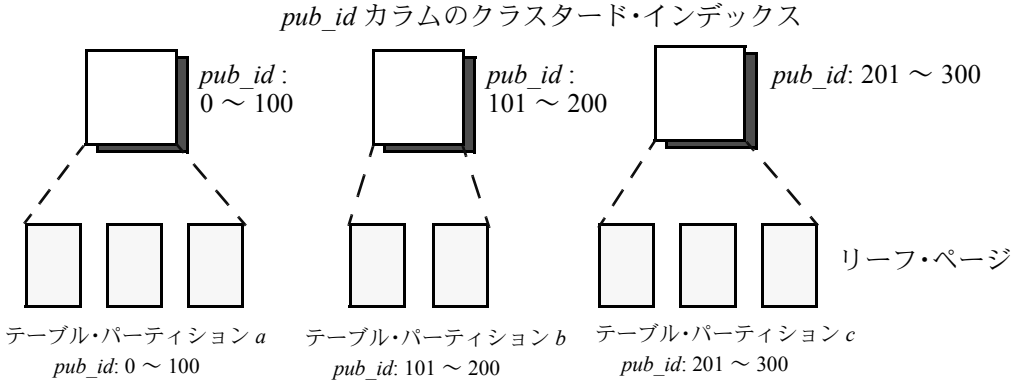
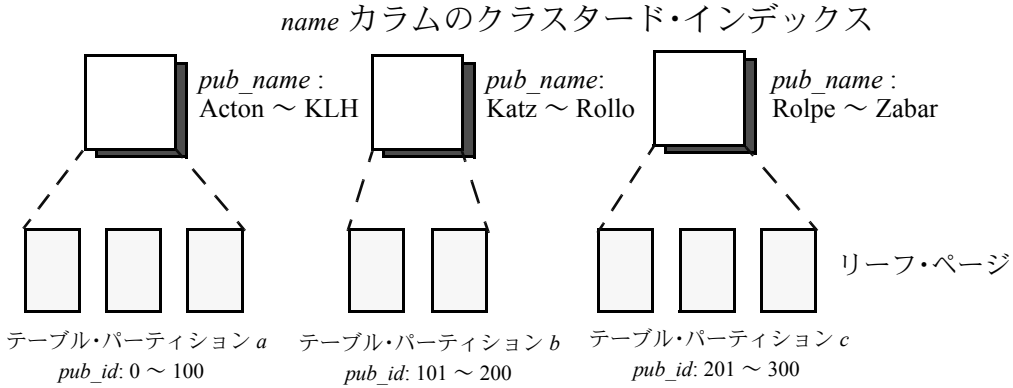


図 10-7 の例では、インデックスは pub_name カラムに対して作成されています。ユニーク性は強制できません。「ユニーク・インデックスの保証」(362 ページ) を参照してください。

この例を範囲分割された publishers テーブル上に作成するには、次のように入力します。

```
create clustered index publish7_idx
    on publishers(pub_name)
    local index p1, p2, p3
```

図 10-7: ユニークでないローカル・クラスタード・インデックス



ローカル・ノンクラスタード・インデックス

ローカル・ノンクラスタード・インデックスは、インデックスを使用可能な任意のカラムのセットに定義できます。

「分割されたテーブルのグローバル・ノンクラスタード・インデックス」(358 ページ) と同様に、pub_id カラムで範囲分割した publishers テーブルを使用して、分割されたノンクラスタード・インデックスを pub_id カラムと city カラムに対して作成します。

```
create nonclustered index publish8_idx (A)
    on publishers(pub_id, city)
    local index p1, p2, p3
```

分割されたノンクラスタード・インデックスを city カラムに作成することもできます。

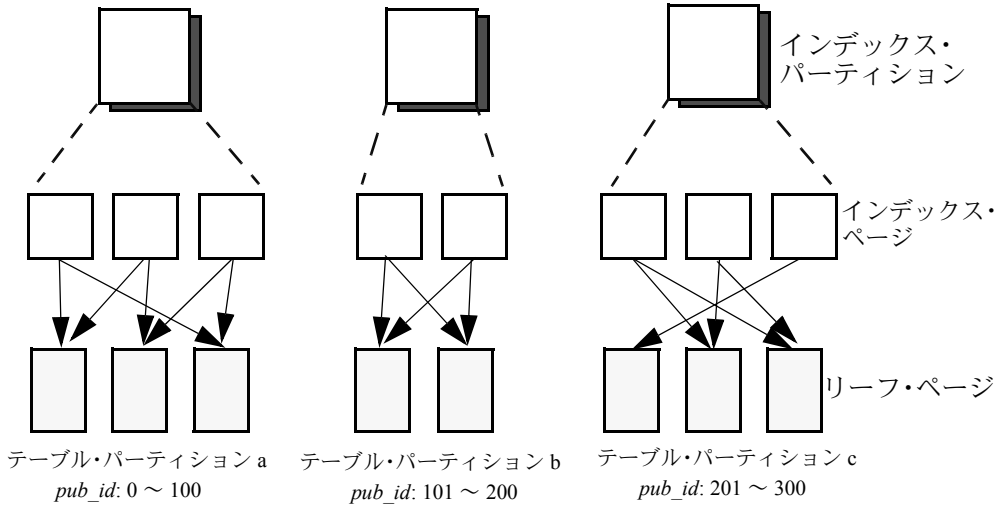
```
create nonclustered index publish9_idx (B)
    on publishers(city)
    local index p1, p2, p3
```

図 10-8 は、ノンクラスタード・ローカル・インデックスの例を示しています。各例の図はまったく同じですが、例 A ではユニーク性を強制でき、例 B ではユニーク性を強制できません。「ユニーク・インデックスの保証」(362 ページ) を参照してください。

図 10-8: ローカル・ノンクラスタード・インデックス

A. 左にプレフィクスが付いたインデックス:
インデックス・カラム: *pub_id*、*city*
インデックス分割キー: *pub_id*

B. プレフィクスが付かないインデックス:
インデックス・カラム: *city*
インデックス分割キー: *pub_id*



ユニーク・インデックスの保証

ユニーク・インデックスを使用すると、null 値も含めて同じインデックス値を持つ 2 つのローは許可されません。既にデータが存在する場合は、インデックスの作成時に、または insert コマンドや update コマンドによってデータを追加または修正するたびに、システムは重複値がないかどうかをチェックします。ユニーク・インデックスの作成の詳細については、「第 13 章 テーブルのインデックスの作成」を参照してください。

グローバル・インデックスは分割されていないため、unique キーワードを使用して簡単にユニーク性を強制できます。一方、ローカル・インデックスは分割されているため、ユニーク性を強制するには追加の制約が必要になります。

ローカル・インデックスのユニーク性を強制するには、分割キーが次の条件を満たす必要があります。

- インデックス・キーのサブセットである。
- インデックス・キーと同じ順序である。

たとえば、以下の例ではユニーク性を強制できます。

- インデックス・キーが column1 に設定されたローカル・インデックスを持つ、column1 でハッシュ、リスト、または範囲により分割されたテーブル。
- インデックス・キーが column1 と column2 に設定されたローカル・インデックスを持つ、column1 でハッシュ、リスト、または範囲により分割されたテーブル。図 10-8 の例 A を参照。
- column1 と column3 でハッシュ、リスト、または範囲により分割されたテーブル。ローカル・インデックスは、次のいずれかのインデックス・キーを持ちます。
  - column1、column3
  - column1、column2、column3
  - column0、column1、column3、column4

  column3 または column1、column3 をインデックス・キーとして使用するインデックスは、ユニーク性を強制できない。

ローカル・インデックスを持つ、ラウンドロビン方式で分割されたテーブルにはユニーク性を強制できません。

パーティションの作成と管理

ライセンスされたサイトでセマンティック分割を有効にするには、次のように入力します。

```
sp_configure 'enable semantic partitioning', 1
```

ラウンドロビン分割は常に使用でき、enable semantic partitioning の値には影響されません。

以下のような通常使用する管理操作や保守操作を実行する場合は、セマンティック分割を有効にしてください。

- テーブルの作成とトランケート (create table、truncate table)
- ロックの変更またはスキーマの修正のためのテーブルの変更 (alter table)
- インデックスの作成 (create index)
- 統計情報の更新 (update statistics)
- クラスタード・インデックスに従ったテーブル・ページの再編成と領域の最適な使用 (reorg rebuild)

分割タスク

テーブルやインデックスを分割する前に、パーティションとして使用するディスク・デバイスとセグメント、またはその他の記憶デバイスを準備しておく必要があります。

複数のパーティションを 1 つのセグメントに割り当てることができます。ただし、各パーティションは 1 つのセグメントだけに割り当てることができます。デバイスを個々のセグメントにバインドし、各セグメントに 1 つのパーティションを割り当てることにより、並列化および分割の利点を最大限に活用できます。

分割タスクの一般的な順序は次のとおりです。

1 disk init を使用して、新しいデータベース・デバイスを初期化します。disk init は、物理ディスク・デバイスまたはオペレーティング・システム・ファイルを論理データベース・デバイス名にマッピングします。次に例を示します。

```
use master
disk init
name ="pubs_dev1",
physname = "SYB_DEV01/pubs_dev",
size = "50M"
```

『システム管理ガイド 第 1 巻』の「第 7 章 データベース・デバイスの初期化」を参照してください。

2 alter database を使用して、分割対象のテーブルまたはインデックスを含むデータベースに新しいデバイスを割り当てます。次に例を示します。

```
use master
alter database pubs2 on pubs_dev1
```

3 (オプション) sp_addsegment を使用してデータベースのセグメントを定義します。この例は、pubs_dev1 と同様の方法で pubs_dev2、pubs_dev3、pubs_dev4 が作成されていることを前提としています。

```
use pubs2
sp_addsegment seg1, pubs2, pubs_dev1
sp_addsegment seg2, pubs2, pubs_dev2
sp_addsegment seg3, pubs2, pubs_dev3
sp_addsegment seg4, pubs2, pubs_dev4
```

4 すべてのインデックスを分割対象のテーブルから削除します。次に例を示します。

```
use pubs2
drop index salesdetail.titleidind,
    salesdetail.salesdetailind
```

5　sp_dboption を使用して、新しいパーティションへのテーブル・データまたはインデックス・データのバルク・コピーを有効にします。次に例を示します。

```
use master
sp_dboption pubs2,"select into", true
```

6　alter table を使用してテーブルを再分割するか、create table を使用してパーティションを持つ新しいテーブルを作成します。または、create index を使用して新しい、分割されたインデックスを作成するか、select into を使用して、新しい、分割されたテーブルを既存のテーブルから作成します。

たとえば、pubs2 の salesdetail テーブルを再分割するには、次のように入力します。

```
use pubs2
alter table salesdetail partition by range (qty)
    (smsales values <= (1000) on seg1,
    medsales values <= (5000) on seg2,
    lgsales values <= (10000) on seg3)
```

7　分割されたテーブルのインデックスを再作成します。たとえば、salesdetail テーブルのインデックスを再作成するには、次のように入力します。

```
use pubs2
create nonclustered index titleidind
    on salesdetail (title_id)
create nonclustered index salesdetailind
    on salesdetail (stor_id)
```

データ・パーティションの作成

この項では、create table を使用して、範囲、ハッシュ、リスト、ラウンドロビン方式で分割されたテーブルを作成する方法について説明します。『リファレンス・マニュアル：コマンド』を参照してください。

範囲分割されたテーブルの作成

この例では、fictionsales という名前の範囲により分割されたテーブルを作成します。このテーブルは、年度の各四半期に対応する 4 つのパーティションを持ちます。最高のパフォーマンスを得るため、各パーティションは独立したセグメントに配置します。

```
create table fictionsales
    (store_id int not null,
    order_num int not null,
    date datetime not null)
partition by range (date)
    (q1 values <= ("3/31/2004") on seg1,
```

```
    q2 values <= ("6/30/2004") on seg2,
    q3 values <= ("9/30/2004") on seg3,
    q4 values <= ("12/31/2004") on seg4)
```

分割キーのカラムは date です。q1 パーティションは seg1 上に配置され、date の値が 3/31/2004 までのすべてのローを含みます。q2 パーティションは seg2 上に配置され、日付値が 4/1/2004 ～ 6/30/2004 のすべてのローを含みます。q3 と q4 も同様に分割されています。

"12/31/2004" よりも後の date 値を挿入しようとするとエラーが発生し、挿入は失敗します。このように、範囲条件はテーブルの検査制約として動作します。これは、テーブルに挿入できる行を制限することによって行われます。

データ型の最大値までのすべての値が含まれることを確認するには、最後に作成されたパーティションの上限値として MAX キーワードを使用します。次に例を示します。

```
create table pb_fictionsales
    (store_id int not null,
    order_num int not null,
    date datetime not null)
partition by range (order_num)
    (low values <= (1000) on seg1,
    mid values <= (5000) on seg2,
    high values <= (MAX) on seg3)
```

範囲分割されたテーブルの分割キーと境界値に関する制約

パーティションの境界値は、パーティション作成時の順序に基づいた昇順である必要があります。つまり、2 番目のパーティションの上限値は最初のパーティションの上限値よりも大きい必要があり、その後のパーティションについても同様です。

また、パーティションの境界値は、対応する分割キー・カラムのデータ型と互換性がある必要があります。たとえば、varchar は char と互換性があります。境界値のデータ型が対応する分割キー・カラムのデータ型と異なる場合、境界値は分割キー・カラムのデータ型に変換されます。ただし、次のような例外があります。

- 明示的変換は指定できません。次の例は、varchar から int への不正な変換を試行します。

  ```
  create table employees(emp_names varchar(20))
          partition by range(emp_name)
      (p1 values <= (1),
      p2 values <= (10))
  ```

- データの消失につながる暗黙的変換は指定できません。次の例では、Adaptive Server が境界値を integer 値に変換する場合、丸め条件によってはデータが消失する可能性があります。パーティションの境界値は、分割キーのデータ型と互換性がありません。

```
create table emp_id (id int)
        partition by range(id)
    (p1 values <= (10.5),
    p2 values <= (100.5))
```

次の例では、パーティションの境界値と分割キーのデータ型は互換性があります。境界値は、float 値に直接変換されます。丸めは必要なく、変換はサポートされます。

```
create table id_emp (id float)
        partition by range(id)
    (p1 values <= (10),
    p2 values <= (100))
```

- binary 以外のデータ型から binary データ型への変換は指定できません。たとえば、次のような変換は認められません。

```
create table newemp (name binary)
        partition by range(name)
    (p1 values <= (“Maarten”),
    p2 values <= (“Zymmerman”)
```

ハッシュ分割されたテーブルの作成

次の例は、3 つのハッシュ・パーティションを持つテーブルを作成します。

```
create table mysalesdetail
    (store_id char(4) not null,
    ord_num varchar(20) not null,
    title_id tid not null,
    qty smallint not null,
    discount float not null)
partition by hash (ord_num)
(p1 on seg1, p2 on seg2, p3 on seg3)
```

ハッシュ分割されたテーブルは、作成と管理が簡単です。Adaptive Server はハッシュ関数を選択し、パーティション間にローを均等に分散しようとします。

ハッシュ分割では、すべてのローはいずれかのパーティションに属することが保証されます。挿入や更新でパーティションの検出に失敗する可能性はありません。これは、範囲またはリストにより分割されたテーブルには当てはまりません。

リスト分割されたテーブルの作成

リスト分割は、特定のパーティションへの個々のローのマッピング方法を制御します。リスト分割は順序付けられていないため、小さいカーディナリティ値に最適です。各パーティションの値リストは 1 つ以上の値を持つ必要があり、同じ値が複数のリストに表示されることはありません。

次の例は、2 つのリスト・パーティションを持つテーブルを作成します。

```
create table my_publishers
 (pub_id char (4) not null,
pub_name varchar(40) null,
city varchar(20) null,
state char(2) null)
partition by list (state)
(west values ('CA', 'OR', 'WA') on seg1,
east value ( 'NY', 'NJ') on seg2)
```

リストにない値を state カラムに持つローを挿入しようとすると失敗します。同様に、リストにないキー・カラム値を持つ既存の行を更新しようとすると失敗します。範囲分割されたテーブルの場合と同様、各リストの値はテーブル全体の検査制約として機能します。

ラウンドロビン方式で分割されたテーブルの作成

分割基準が使用されないため、この分割方式はランダムです。ラウンドロビン方式で分割されたテーブルは分割キーを持ちません。

次の例は、ラウンドロビン分割を指定しています。

```
create table currentpublishers
 (pub_id char (4) not null,
pub_name varchar(40) null,
city varchar(20) null,
state char(2) null)
partition by roundrobin 3 on (seg1)
```

セマンティック分割がライセンスされているまたは設定されているかどうかに関係なく、ラウンドロビン方式で分割されたテーブルにはすべてのパーティション対応ユーティリティと管理タスクを使用できます。

分割されたインデックスの作成

この項では、create index を使用して分割されたインデックスを作成する方法について説明します。『リファレンス・マニュアル：コマンド』を参照してください。

インデックスは、逐次モードまたは並列モードで作成できます。しかし、ラウンドロビン方式で分割されたテーブルのグローバル・インデックスは、並列モードでのみ作成できます。『パフォーマンス&チューニング・シリーズ：クエリ処理と抽象プラン』の「第 7 章 最適化の制御」を参照してください。

グローバル・インデックスの作成

グローバル・クラスタード・インデックスは、ラウンドロビン方式で分割されたテーブルに対してのみ作成できます。Adaptive Server では、すべての種類の分割されたテーブルに対して、分割されていないグローバル・ノンクラスタード・インデックスがサポートされています。

分割されたテーブルのクラスタードおよびノンクラスタード・グローバル・インデックスは、バージョン 12.5.x 以前の Adaptive Server でサポートされている構文を使用して作成できます。

グローバル・インデックスの作成

分割されたテーブルのインデックスを作成すると、次の場合に自動的にグローバル・インデックスが作成されます。

- local index キーワードを含めずに、分割されたテーブルのノンクラスタード・インデックスを作成します。たとえば、「ハッシュ分割されたテーブルの作成」(367 ページ) で説明されているハッシュ分割されたテーブル mysalesdetail の場合は、次のように入力します。

  ```
  create nonclustered index ord_idx on mysalesdetail (au_id)
  ```

- local index キーワードを含めずに、ラウンドロビン方式で分割されたテーブルのクラスタード・インデックスを作成します。たとえば、「ラウンドロビン方式で分割されたテーブルの作成」(368 ページ) で説明されているテーブル currentpublishers の場合は、次のように入力します。

  ```
  create clustered index pub_idx on currentpublishers
  ```

ローカル・インデックスの作成

Adaptive Server では、すべての種類の分割されたテーブルに対して、ローカル・クラスタード・インデックスとローカル・ノンクラスタード・インデックスがサポートされています。ローカル・インデックスは、ベース・テーブルのパーティション方式、パーティション・カラム、およびパーティション境界を継承します。

範囲、ハッシュ、リストにより分割されたテーブルの場合、create index 文にキーワード local index が含まれるかどうかに関係なく、常にローカル・クラスタード・インデックスが作成されます。

次の例は、分割された mysalesdetail テーブルにローカルのクラスタード・インデックスを作成します (「ハッシュ分割されたテーブルの作成」(367 ページ) を参照)。クラスタード・インデックスでは、インデックス・ローとデータ・ローの物理的な順序が同じである必要があります。クラスタード・インデックスは、1 つのテーブルに対して 1 つだけ作成できます。

```
create clustered index clust_idx
    on mysalesdetail(ord_num) local index
```

次の例は、分割された mysalesdetail テーブルのローカル・ノンクラスタード・インデックスを作成します。このインデックスは、title_id で分割されます。1 つのテーブルにつき最大 249 のノンクラスタード・インデックスを作成できます。

```
create nonclustered index nonclust_idx
       on mysalesdetail(title_id)
    local index p1 on seg1, p2 on seg2, p3 on seg3
```

分割されたテーブルへのクラスタード・インデックスの作成

次に示す条件が満たされていれば、分割されたテーブルに対してクラスタード・インデックスを作成できます。

- select into/bulkcopy/pllsort データベース・オプションが true に設定されている。
- テーブルの分割部分の数と同数のワーカー・スレッドを使用できる。

注意 ラウンドロビン方式で分割されたテーブルのグローバル・インデックスを作成する前に、サーバが並列実行されるように設定されていることを確認してください。

リカバリを高速に行うため、クラスタード・インデックスを作成してからデータベースをダンプしてください。

分割されたテーブルにクラスタード・インデックスを作成する前に、システム管理者またはデータベース管理者に確認してください。

分割されたテーブルの既存のテーブルからの作成

分割されたテーブルを既存のテーブルから作成するには、select into コマンドを使用します。select と into 句を使用して、範囲、ハッシュ、リスト、またはラウンドロビン方式で分割されたテーブルを作成できます。
選択元のテーブルは、分割されていても分割されていなくてもかまいません。『リファレンス・マニュアル：コマンド』を参照してください。

注意 アプリケーション内で select と into 句を使用すると、分割されたテンポラリ・テーブルを tempdb に作成できます。

たとえば、分割された sales_report テーブルを salesdetail テーブルから作成するには、次のように入力します。

```
select * into sales_report partition by range (qty)
(smallorder values <= (500) on seg1,
bigorder values <= (5000) on seg2)
from salesdetail
```

データ・パーティションの変更

alter table コマンドは以下の目的で使用できます。

- 分割されていないテーブルを複数パーティションのテーブルに変更する。
- 1 つ以上のパーティションをリストまたは範囲により分割されたテーブルに追加する。
- 異なる分割方式でテーブルを再分割する。
- 異なる分割キーまたは境界値でテーブルを再分割する。
- 異なるパーティション数でテーブルを再分割する。
- テーブルを再分割して、異なるセグメントにパーティションを割り当てる。

『リファレンス・マニュアル：コマンド』を参照してください。

❖ **テーブルの再分割**

テーブルを再分割するための一般的な手順は次のとおりです。

1　再分割プロセス中に分割キーまたはパーティション方式を変更する場合は、テーブルのすべてのインデックスを削除します。

2　alter table を使用してテーブルを再分割します。

3　再分割プロセス中に分割キーまたはタイプを変更した場合は、テーブルのインデックスを再作成します。

分割されていないテーブルから分割されたテーブルへの変更

次の例は、分割されていない titles テーブルを、3 つの範囲パーティションを持つテーブルに変更します。

```
alter table titles partition by range (total_sales)
(smallsales values <= (500) on seg1,
mediumsales values <= (5000) on seg2,
bigsales values <= (25000) on seg3)
```

分割されたテーブルへのパーティションの追加

パーティションは、リストまたは範囲により分割されたテーブルに追加できますが、ハッシュまたはラウンドロビン方式で分割されたテーブルには追加できません。次の例は、既存の分割キー・カラムを使用して、範囲分割されたテーブルに新しいパーティションを追加します。

```
alter table titles add partition
(vbigsales values <= (40000) on seg4)
```

注意 パーティションは、既存の範囲ベースのパーティションの最上位にしか追加できません。パーティションに対して values <= (MAX) を定義している場合、新しいパーティションは追加できません。

リストまたは範囲分割されたテーブルにパーティションを追加する場合、データはコピーされません。新しく作成されるパーティションは空です。

分割方式の変更

次の titles テーブルは、「分割されたテーブルへのパーティションの追加」(371 ページ) で範囲分割されたものです。次の例では、title_id カラムで、ハッシュにより titles を再分割します。

```
alter table titles partition by hash(title_id)
    3 on (seg1, seg2, seg3)
```

もう一度、total_sales カラムの範囲により titles を再分割します。

```
alter table titles partition
        by range (total_sales)
    (smallsales values <= (500) on seg1,
    mediumsales values <= (5000) on seg2,
    bigsales values <= (25000) on seg3)
```

注意 分割方式を変更する前に、すべてのインデックスを削除する必要があります。

分割キーの変更

次の titles テーブルは、「分割方式の変更」(372 ページ) で total_sales カラムの範囲により再分割されたものです。この例では、分割キーを変更しますが、分割方式は変更しません。

```
alter table titles partition by range(pubdate)
    (q1 values <= ("3/31/2006"),
    q2 values <= ("6/30/2006"),
    q3 values <= ("9/30/2006"),
    q1 values <= ("12/31/2006"))
```

注意 分割キーを変更する前に、すべてのインデックスを削除する必要があります。

ラウンドロビン方式で分割されたテーブルの分割解除

すべてのパーティションが同じセグメントにあり、テーブルのインデックスが存在しない場合は、alter table と unpartition 句を使用して、分割されたラウンドロビン・テーブルから分割されていないラウンドロビン・テーブルを作成できます。この機能は、最終的には分割されていないテーブルとして使用するテーブルに大量のデータをロードする場合に役立ちます。「パーティションを使用したテーブル・データのロード」(377 ページ) を参照してください。

partition パラメータの使用

partition *number_of_partitions* パラメータを使用して、分割されていないラウンドロビン・テーブルを、指定した数のパーティションを持つラウンドロビン方式で分割されたテーブルに変更できます。Adaptive Server では、既存のデータはすべて最初のパーティションに配置されます。残りのパーティションは空のパーティションとして作成され、最初の既存パーティションと同じセグメントに配置されます。それ以降に挿入されるデータは、ラウンドロビン方式により、すべてのパーティションに分配されます。

ローカル・インデックスが初期パーティションに存在する場合は、空のローカル・インデックスが新しいパーティションに構築されます。テーブルの作成時にセグメントを宣言した場合、新しいパーティションはそのセグメントに配置されます。それ以外の場合、パーティションは、テーブルおよびインデックス・レベルで指定されたデフォルトのセグメントに配置されます。

たとえば、次のように pubs2 の discounts テーブルに partition *number_of_partitions* を使用して、3 つのラウンドロビン・パーティションを作成できます。

```
alter table discounts partition 3
```

注意　alter table における partition 句の使用は、ラウンドロビン・パーティションを作成するためだけにサポートされています。その他の種類のパーティションの作成ではサポートされていません。

分割キー・カラムの変更

分割キー・カラムを変更する場合は注意が必要です。以下のルールが適用されます。

- 分割キーの一部となっているカラムは削除できません。分割キーの一部ではないカラムは削除できます。
- 範囲分割されたテーブルの分割キーの一部となっているカラムのデータ型を変更すると、そのパーティションの境界値が新しいデータ型に変換されます。ただし、以下の例外があります。
    - 明示的変換

- データの消失につながる暗黙的変換
- binary 以外のデータ型から binary データ型への変換

サポートされていない変換の詳細と例については、「範囲分割されたテーブルの分割キーと境界値に関する制約」(366 ページ) を参照してください。

分割キー・カラムのデータ型を変更すると、データが複数のパーティションに再分散されることがあります。

- 範囲パーティションの場合 – パーティション境界値に近い分割キー値が存在すると、データ型変換によってローが別のパーティションに移動されることがあります。

  たとえば、分割キーの元のデータ型が float で、それを integer に変換するとします。パーティション境界値は、`p1 values <= (5)`、`p2 values <= (10)` になります。分割キー 5.5 を持つローは 5 に変換され、ローは p2 から p1 へ移動されます。

- 範囲パーティションの場合 – 分割キーのデータ型が変更されたためにソート順が変更されると、新しいソート順に基づいてすべてのデータ・ローが再分割されます。たとえば、分割キーのデータ型が varchar から datetime へ変更されると、ソート順が変更されます。

  分割キー・カラムのデータ型を変更しようとすると alter table は失敗し、変換後に新しい境界値で必要な昇順が保持されなかったり、すべてのローが新しいパーティションに収まらなかったりします。

  詳細については、『システム管理ガイド 第 1 巻』の「第 9 章 文字セット、ソート順、言語の設定」の「疑わしいパーティションの処理」の項を参照してください。

- ハッシュ・パーティションの場合 – データ値と分割キーのデータ型の記憶サイズを使用してハッシュ値が生成されます。その結果、ハッシュ分割キーのデータ型を変更すると、データが再分配される場合があります。

パーティションの設定

パーティションを設定すると、パフォーマンスが向上する可能性があります。パーティションの設定パラメータは次のとおりです。

- number of open partitions – Adaptive Server が一度にアクセスできるパーティションの数を指定します。デフォルト値は 500 です。
- partition spinlock ratio – オープン・パーティションへの同時アクセスに対する保護を提供するスピンロックの数を指定します。デフォルト値は 10 です。

『システム管理ガイド 第 1 巻』の「第 5 章 設定パラメータ」を参照してください。

分割されたテーブルでの更新、削除、挿入

分割されたテーブルのデータを更新、挿入、削除するための構文は、分割されていないテーブルの場合と同じです。update、insert、delete 文にはパーティションを指定できません。

分割されたテーブルでは、データはパーティション上に配置され、テーブルはパーティションを論理的に結合したものになります。特定のデータ・ローが格納されている正確なパーティションは、ユーザに対して透過的です。Adaptive Server は、アクセスされるパーティションを、内部論理とテーブルの分割方式の組み合わせによって決定します。

テーブルのパーティションの条件を満たさない行を挿入しようとするトランザクションはアボートします。ラウンドロビンまたはハッシュ方式で分割されたテーブルでは、すべてのローが条件を満たします。範囲またはリストにより分割されたテーブルでは、分割基準を満たすローだけが条件を満たします。

- 範囲分割されたテーブルの場合 － MAX キーワードが指定されていないときは、テーブルに定義された範囲の上限を超える値を持つデータ行を挿入するとアボートします。MAX キーワードが指定されている場合は、すべてのローが上限の条件を満たします。
- リスト分割されたテーブル － 分割基準と一致しないパーティション・カラム値を持つデータ・ローは挿入できません。

データ・ローの分割キー・カラムが更新された結果、キー・カラム値がいずれのパーティションの分割基準も満たさなくなった場合、更新はアボートします。「分割キー・カラムの値の更新」(375 ページ) を参照してください。

分割キー・カラムの値の更新

セマンティック分割テーブルの場合、分割キー・カラムの値を更新すると、データ・ローが別のパーティションに移動する可能性があります。

データ・ローを別のパーティションに移動する必要がある場合、Adaptive Server は遅延モードで分割キー・カラムを更新します。遅延更新は 2 段階で行われます。つまり、ローは元のパーティションから削除されてから、新しいパーティションに挿入されます。

データオンリーロック・テーブルに対してこのような操作を実行すると、ロー ID (RID) が変更され、スキャン異常が発生する可能性があります。たとえば、カラム a で範囲分割されたテーブルを作成するとします。

```
create table test_table (a int) partition by range (a)
    (partition1 <= (1),
    partition2 <= (10))
```

このテーブルは、partition2 に単一のローを持ちます。分割キー・カラムの値は 2 です。partition1 は空です。次のようなトランザクションがあるとします。

```
Transaction T1:
      begin tran
    go
    update table set a = 0
    go
Transaction T2:
    select count(*) from table isolation level 1
    go
```

T1 を更新すると、単一のローが partition2 から削除され、partition1 に挿入されます。ただし、delete と insert は、いずれもこの時点ではコミットされていません。したがって、T2 の select count(*) は、コミットされていない partition1 の insert でブロックされません。その代わり、コミットされていない partition2 の delete でブロックされます。T1 がコミットしても、T2 にはコミットされた delete が見えないため、カウント値としてゼロ (0) を返します。

この動作は、パーティションを使用しないデータオンリーロック・テーブルに対する inserts および deletes で見られます。別のパーティションにローが移動するように分割キーの値が更新される場合は、update だけに発生します。詳細については、『パフォーマンス&チューニング・シリーズ：物理データベースのチューニング』の「第 1 章 データの物理的配置の制御」、および『パフォーマンス&チューニング・シリーズ：ロックと同時実行制御』の「第 5 章 インデックス」を参照してください。

パーティションに関する情報の表示

パーティションに関する情報を表示するには、sp_helpartition を使用します。たとえば、publishers の p1 パーティションに関する情報を表示するには、次のように入力します。

```
sp_helpartition publishers, null, p1
```

『リファレンス・マニュアル：プロシージャ』を参照してください。

関数の使用

パーティション情報を表示するために使用できる関数はいくつかあります。構文と使用法の詳細については、『リファレンス・マニュアル：ビルディング・ブロック』を参照してください。

- data_pages － テーブル、インデックス、パーティションが使用しているページ数を返します。
- reserved_pages － テーブル、インデックス、パーティション用に予約されているページ数を返します。
- row_count － テーブルまたはパーティション内のローの数を予測します。

- used_pages − テーブル、インデックス、パーティションが使用しているページ数を返します。data_pages とは異なり、used_pages は、内部構造に使用されているページ数を含みます。
- partition_id − 指定されたインデックスに対して、指定されたパーティションのパーティション ID を返します。
- partition_name − 指定されたインデックス ID およびパーティション ID に対応するパーティション名を返します。

例

次の例は、指定されたデータベースの、ID が 31000114 のオブジェクトが使用しているページ数を返します。ページ数には、インデックスのページ数も含みます。

```
data_pages(5, 31000114)
```

次の例は、testtable_ptn1 パーティションに対応するパーティション ID を返します。

```
select partition_id("testtable" testtable_ptn1")
```

次の例は、インデックス ID が 0 のベース・テーブルに属する、パーティション ID が 1111111111 のパーティションの名前を返します。

```
select partition_name(0, 1111111111)
```

パーティションのトランケート

他のパーティションの情報に影響を与えずに、パーティション内のすべての情報を削除できます。たとえば、fictionsales テーブルの q1 パーティションと q2 パーティションからすべてのローを削除するには、次のように入力します。

```
truncate table fictionsales partition q1
truncate table fictionsales partition q2
```

『リファレンス・マニュアル：コマンド』を参照してください。

パーティションを使用したテーブル・データのロード

テーブルを最終的には分割されていないテーブルとして使用する場合であっても、分割を利用することで、大量のテーブル・データのロードを効率化できます。

ラウンドロビン分割方式を使用して、すべてのパーティションを同じセグメントに配置します。

1 空のテーブルを作成し、*n* 個に分割します。

```
create table currentpublishers
(pub_id char (4) not null,
pub_name varchar(40) null,
city varchar(20) null,
state char(2) null)
partition by roundrobin 3 on (seg1)
```

2 *partition_id* オプションを使用して bcp in を実行します。事前にソートされたデータを各パーティションにコピーします。たとえば、currentpublishers の最初のパーティションに *datafile1.dat* をコピーするには、次のように入力します。

```
bcp pubs2..currentpublishers:1 in datafile1.dat
```

3 テーブルの分割を解除します。

```
alter table currentpublishers unpartition
```

4 クラスタード・インデックスを作成します。

```
create clustered index pubnameind
    on currentpublishers(pub_name)
    with sorted_data
```

パーティションが作成されると、Adaptive Server は syspartitions テーブルの各パーティションにエントリを配置します。bcp in を *partition_id* オプションとともに使用すると、syspartitions にリストされた順序で各パーティションにデータがロードされます。この順序を維持するため、テーブルを分割解除してから、クラスタード・インデックスを作成しました。

分割統計値の更新

Adaptive Server のクエリ・プロセッサは、クエリ内でテーブル、インデックス、パーティション、カラムについての統計値を使用してクエリのコストを見積もります。クエリ・プロセッサは、最もコストが低いと判断したアクセス・メソッドを選択します。ただし、そのためには、統計値が正確である必要があります。

一部の統計値はクエリ処理中に更新されます。その他の統計値は、update statistics コマンドを実行したときか、インデックスを作成したときにだけ更新されます。

update statistics により、Adaptive Server は、パーティションのローカル・インデックスの主要属性ごとにヒストグラムを作成し、複合属性の密度を作成して最適な決定を下します。update statistics コマンドは、分割されたテーブルで大量のデータが追加、変更、削除された場合に使用します。

update statistics および delete statistics を発行するパーミッションは、デフォルトではテーブル所有者に与えられ、他のユーザには譲渡できません。update statistics コマンドを使用すると、個々のデータ・パーティションおよびインデックス・パーティションを更新できます。パーティションに関する情報を取得する update statistics コマンドは次のとおりです。

- update statistics
- update table statistics
- update all statistics
- update index statistics
- delete statistics

たとえば、「分割されていないテーブルから分割されたテーブルへの変更」(371 ページ) で作成した titles テーブルの smallvalues パーティションの統計値を更新するには、次のように入力します。

```
update statistics titles partition smallvalues
```

『リファレンス・マニュアル：コマンド』を参照してください。

第 11 章

仮想ハッシュ・テーブル

注意 仮想ハッシュ・テーブルは、IBM Linux pSeries および Linux AMD64 でのみ使用できます。

ハッシュベース・インデックス・スキャンは、ノンクラスタード・インデックスまたはデータオンリーロック・テーブルのクラスタード・インデックスで実行できます。スキャン時に、各ワーカー・プロセスは、上位レベルのインデックスを操作し、インデックスのリーフレベル・ページを読み込みます。次に、各ワーカー・プロセスが別個のハッシュ・テーブル内のデータ・ページ ID または値に基づいてハッシュを行い、どのデータ・ページまたはデータ・ローを処理するか決定します。

仮想ハッシュ・テーブルは、別個のハッシュ・テーブルを必要としないため、テーブルを効率的に整理できる方法です。代わりに、ハッシュ・キーを使用してクエリ プロセッサがロー ID (ローの序数に基づく) およびデータの位置を判断できるように、ローを格納します。別個のハッシュ・テーブルを使用して情報を保持しないため、「仮想」ハッシュ・テーブルと呼ばれます。

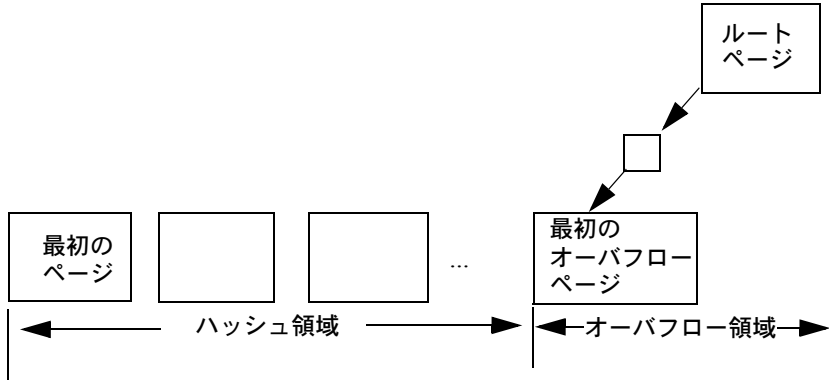
効率の高い CPU 使用率を必要とするシステムには、仮想ハッシュ・テーブルの使用が適しています。

ルックアップに使用されるテーブル、またはロー位置が変化しないテーブルに、クラスタード・インデックスまたはノンクラスタード・インデックスを使用すると大きなコストがかかります。L2 および L3 CPU アーキテクチャは近年進歩しているため、キャッシュを利用して実際の CPU 計算能力の利点を活かす必要があります。キャッシュを利用しない場合、CPU は使用可能なメモリを待機するために不必要なサイクルを消費します。クラスタード・インデックスやノンクラスタード・インデックスの場合、サーバは多くの CPU サイクルを消費するインデックス・レベル検索にアクセスするたびに、ローをミスします。仮想ハッシュ・テーブルは、検索を実行する代わりにハッシュ・キー値を計算することで、ロー位置パターンにアクセスします。

仮想ハッシュ・テーブルの構造

仮想ハッシュ・テーブルには、「ハッシュ」領域と「オーバーフロー」領域の 2 つの領域が含まれます。ハッシュ領域にはハッシュ・ローが格納され、オーバーフロー領域には残りのローが格納されます。オーバーフロー領域には、B ツリー・クラスタード・インデックスを使用した通常のクラスタード・インデックス・スキャンによってアクセスします。

図 11-1: ルート・ページを使用したオーバーフロー領域へのアクセス



仮想ハッシュ・テーブルの最初のデータ・ページ、ルート・ページ、および最初のオーバーフロー・ページは、テーブルの作成時に作成されます。SYSINDEXES.indroot は、オーバーフロー・クラスタード領域のルート・ページです。このページの下の最初のリーフ・ページは、最初のオーバーフロー・ページです。SYSINDEXES.indfirst は、は、最初のデータ・ページを指しているため、テーブル・スキャンはテーブルの最初から開始して、テーブル全体をスキャンします。

仮想ハッシュ・テーブルの作成

仮想ハッシュ・テーブルを作成するには、ハッシュ領域の最大値を指定します。これは、create table の部分構文です。仮想ハッシュ・テーブル用のパラメータは太字で表記されています。

```
create table [database.[owner].]table_name
. . .
        | {unique | primary key}
          using clustered
        (column_name [asc | desc] [{, column_name [asc | desc]}...])=
          (hash_factor [{, hash_factor}...])
                with max num_hash_values key
```

それぞれの意味は、次のとおりです。

- using clustered － 仮想ハッシュ・テーブルを作成することを示します。カラムのリストは、このテーブルのキー・カラムとして扱われます。
- *column_name* [asc | desc] － ローはそのハッシュ関数に基づいて配置されるため、ハッシュ領域に [asc | desc] を使用することはできません。仮想ハッシュ・テーブルのキー カラムに順序を指定した場合、オーバフロー・クラスタード領域でのみ使用されます。
- *hash_factor* － 仮想ハッシュ・テーブルのハッシュ関数に対して必要です。ハッシュ関数の場合、キー・カラムごとにハッシュ係数が必要です。これらの係数はキー値とともに使用され、特定のローにハッシュ値を生成します。
- with max *num_hash_values* key － 使用できるハッシュ値の最大数。このハッシュ関数の出力における上限を定義します。

hash_factor の値の決定

最初のキーのハッシュ係数を 1 に保つことができます。残りのすべてのキー カラムに対するハッシュ係数は、そのハッシュ係数で乗算されたハッシュ領域で許可される以前のキーの最大値よりも大きくなります。

Adaptive Server では、ページのローをより少なくするために、最初のキー・カラムに対するハッシュ係数が 1 より大きいテーブルを許可しています。たとえば、テーブルに最初のキー・カラムに対してハッシュ係数 5 がある場合、ページの各ローの後、次の 4 つのローの領域は空のままです。これをサポートするために、Adaptive Server にはテーブル領域の 5 倍が必要です。

キー・カラムの値が、次のキー・カラムのハッシュ係数と等しいか大きい場合は、ハッシュ領域で衝突が発生しないように、現在のローがオーバフロー・クラスタード領域に挿入されます。

たとえば、t は、id および age キー・カラムと対応するハッシュ係数 (10,1) を持つ仮想ハッシュ・テーブルです。(5, 5) および (2, 35) ローに対するハッシュ値が 55 であるため、ハッシュが衝突する可能性があります。

ただし、値 35 は 10 (次のキー・カラム id のハッシュ係数) 以上であるため、ハッシュ領域で衝突しないように、オーバフロー・クラスタード領域に 2 番目のローが格納されます。

また、u は、プライマリ・インデックスと (id1, id2, id3) = (125, 25, 5) と 200 のうち *max hash_value* のハッシュ係数を持つ仮想ハッシュ・テーブルです。

- ロー (1,1,1) には、ハッシュ値 155 があり、ハッシュ領域に格納されます。
- ロー (2,0,0) には、ハッシュ値 250 があり、オーバフロー・クラスタード領域に格納されます。
- ロー (0,0,6) には、25 以上である 6 x 5 のハッシュ係数があるため、オーバフロー・クラスタード領域に格納されます。
- ロー (0,7,0) には、125 以上である 7 x 25 のハッシュ係数があるため、オーバフロー・クラスタード領域に格納されます。

この例は、ハッシュ領域内のロー数、ローの長さ、およびページあたりのローの数が、ハッシュ領域とオーバーフロー領域のページ・レイアウトにどのように影響するかを示しています。この例では、order_seg セグメントの pubs2 データベースに orders という仮想ハッシュ・テーブルを作成します。

```
create table orders(
id int,
age int,
primary key using clustered (id,age) = (10,1) with max 1000
key)
on order_seg
```

データのレイアウトは次のとおりです。

- order_seg セグメントはページ ID 51200 から始まります。
- 論理ページ・サイズは 2048 バイトです。
- 最初のデータのオブジェクト アロケーション・マップ (OAM) ページの ID は 51201 です。
- 論理ページ・サイズが 2048 バイトの場合、1 ページあたりの最大ロー数は 168 です。
- 行のサイズは 10 です。
- オーバフロー・クラスタード領域のルート・インデックス・ページは 51217 です。

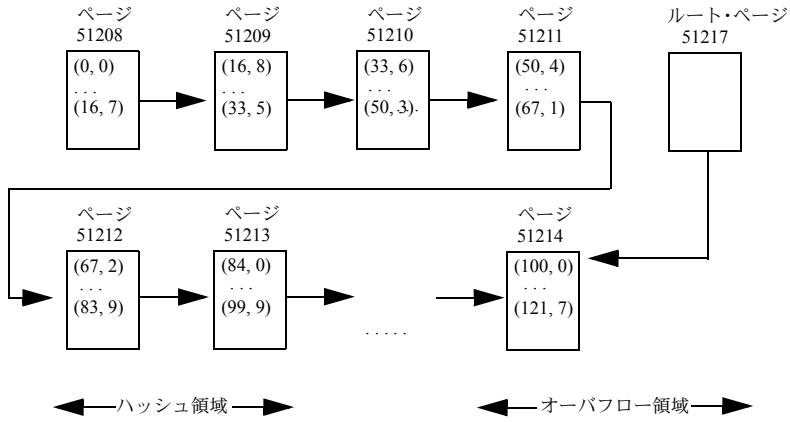
この例では、

- ローのサイズは 10 バイトです。
- ハッシュ領域には (0,0) ～ (99,9) の範囲内のキー値のある 1000 個のローが入ります。

- ハッシュ領域内の総ページ数は 6、ハッシュ領域内のページにはページあたり 168 個のロー、および最大で 1000 個のキー (ceiling(1000/168) = 6) となります。最後のページ (6 ページ目) には、未使用の空領域があります。セグメントが 51200 ページで開始され、最初のエクステントが OAM ページ用に予約されていることを前提とすると、ハッシュ領域内のページは 51208 ～ 51213 ページとなります。

  ハッシュ領域内の最後のページの後のページ (ページ番号 51214) は、クラスタード・インデックスによって制御されるオーバーフロー領域の最初のページとなり、ルート・ページである 51217 は、ページ番号 51214 をポイントします。

図 11-2: データのページ・レイアウト



このページ レイアウトでは、ページあたりのロー数は 168 です。id と age のハッシュ係数はそれぞれ 10 と 1、ハッシュ領域に適合するカラム age は 9 です。ハッシュ領域に適合するキー値の組み合わせ (id と age) の範囲は次のとおりです (合計 1000 キー)。

- (0, 0) ～ (0, 9) の 合計 10
- (1, 0) ～ (1, 9) の 合計 10
- (2, 0) ～ (2, 9) の 合計 10
- . . .
- (99, 0) ～ (99, 9) の 合計 10

これらのキーから、最初の 168 キーである (0, 0) ～ (16, 7) が最初のデータ・ページ 51208 にマップされます。次の 168 キーである (16, 8) ～ (33, 5) は次のデータ・ページ 51209 に、というように続きます。

仮想ハッシュ・テーブルの制限

仮想ハッシュ・テーブルには次の制限があります。

- truncate table はサポートされていません。代わりに delete from *table_name* を使用してください。
- SQL92 では、関連する 2 つの一意性制約が同じキー・カラムを持つことはできないため、Adaptive Server は仮想ハッシュ・テーブルのキー・カラムと同じキー・カラムで、プライマリ・キー制約または一意性キー制約をサポートしません。
- テーブルを作成した後で仮想ハッシュ・クラスタード・インデックスを作成することはできないので、仮想ハッシュ・クラスタード・インデックスを削除することもできません。
- 仮想ハッシュ・テーブルは、排他セグメント上で作成する必要があります。仮想ハッシュ・テーブルを作成するためにセグメントに割り当てるディスク・デバイスは、他のセグメントと共有できません。言い換えると、まず特殊なデバイスを作成してから、そのデバイスに排他セグメントを作成する必要があります。
- 仮想ハッシュ・テーブルにはユニークなローが必要です。仮想ハッシュ・テーブルで複数のローが同じキー・カラム値を持つことはできません。Adaptive Server では、あるローをハッシュ領域に保持し、同じキー・カラム値を持つ別のローをオーバフロー・クラスタード領域に保持することができないためです。
- 同じ排他セグメント上に 2 つの仮想ハッシュ・テーブルを作成することはできません。Adaptive Server では、1 データベースあたり 32 個のセグメントをサポートします。3 個のセグメントがデフォルト、システム、およびログの各セグメント用に予約されるので、1 データベースあたりの仮想ハッシュ・テーブルの最大数は 29 です。
- alter table コマンドと drop clustered index コマンドは仮想ハッシュ・テーブルに対して使用できません。
- 仮想ハッシュ・テーブルでは全ページ・ロックを使用する必要があります。
- 仮想ハッシュ・テーブルのキー・カラムとハッシュ係数は int データ型を使用する必要があります。
- text や image カラムを仮想ハッシュ・テーブルに含めることはできません。text や image データ型に基づくデータ型のカラムも含めることはできません。
- 分割された仮想ハッシュ・テーブルは作成できません。

次のような仮想ハッシュ・テーブルは作成しないでください。

- 挿入および更新が頻繁にある。
- 分割されている。

- 頻繁なテーブル・スキャンを使用する。
- ハッシュ領域よりオーバーフロー領域のデータ・ローの方が多い。この場合、仮想ハッシュ・テーブルの代わりに B ツリーを使用してください。

仮想ハッシュ・テーブルをサポートしているコマンド

create table コマンドの変更点については、「仮想ハッシュ・テーブルの作成」(383 ページ) を参照してください。

- dbcc checktable － 通常の検査に加えて、checktable はハッシュ領域のデータ・ページと OAM ページのレイアウトが正しいことを確認します。
  - レイアウトごとに、OAM ページ用に予約されているエクステント内に、データ・ページは割り付けられません。
  - OAM ページは、アロケーション・ユニットの最初のエクステント内のみに割り付けされます。
- dbcc checkstorage － 非ハッシュ・テーブルの最初のデータ・ページ以外のデータ・ページが空の場合、ソフト・フォールトをレポートします。ただし、dbcc checkstorage は仮想ハッシュ・テーブルのハッシュ領域ではこのソフト・フォールトをレポートしません。仮想ハッシュ・テーブルのハッシュ領域のデータ・ページは空でもかまいません。

クエリ・プロセッサのサポート

クエリ・プロセッサは、等号修飾子 (`where id=2` など) を含む検索引数をすべてのキー・カラムに含めた場合のみ、仮想ハッシュ・インデックスを使用します。クエリ・プロセッサが仮想ハッシュ・インデックスを使用する場合、showplan 出力に次のような行が含まれます。

```
Using Virtually Hashed Index.
```

クエリ・プロセッサが仮想ハッシュ・インデックスを選択した場合、インデックス選択出力に次のような行が含まれます。

```
Unique virtually hashed index found, returns 1 row, 1 pages
```

モニタ・カウンタのサポート

am_srch_hashindex − モニタ・カウンタは、仮想ハッシュ・クラスタード・インデックスを使用して Adaptive Server が検索を実行した回数をカウントします。

システム・プロシージャのサポート

これらのシステム・プロシージャは仮想ハッシュ・テーブルをサポートしています。

- sp_addsegment − 既に排他セグメントを持っているデバイス上にセグメントを作成することはできません。
- sp_extendsegment − 既に排他セグメントがあるデバイスのセグメントは拡張できず、別のセグメントがあるデバイスの排他セグメントは拡張できません。
- sp_placeobject − 仮想ハッシュ・テーブルでは sp_placeobject を使用できず、排他セグメントには他のオブジェクトを配置できません。
- sp_chgattribute − 仮想ハッシュ・テーブルの属性を変更できません。
- sp_help − 仮想ハッシュ・テーブルの場合、次の内容を報告します。
  - テーブルが仮想ハッシュされている
  - テーブルの hash_key_factors

    次に例を示します。

```
attribute_class         attribute       int_value
char_value                                comments
---------------------   ---------------  ------------------------
-------------------------------------    -----------
misc table info         hash key factors                     NULL
id:10.0, id2:1.0, max_hash_key=1000.0           NULL
```

第 12 章

ビュー：データへのアクセスの制限

「ビュー」は、データベース内にオブジェクトとして格納される、select 文です。ビューを使用することによって、1 つ以上のテーブルのローまたはカラムのサブセットを表示することができます。ビューは、Transact-SQL 文でその名前を呼び出して使用します。ユーザは、ビューを使用して、特定のデータベースにあるテーブル内の知りたい情報に焦点をあて、単純化し、カスタマイズすることができます。また、ビューは、必要なデータにだけユーザ・アクセスを許可することによってセキュリティ・メカニズムの役割も果たします。

ビューの機能

ビューは、1 つ以上のテーブルのデータを参照する方法の 1 つです。

たとえば、ユタ州特有のプロジェクトについて作業するとします。次のようにして、ユタ州に住む作家だけをリストするビューを作成できます。

```
create view authors_ut
as select * from authors
where state = "UT"
```

authors_ut ビューを表示するには、次のように入力します。

```
select * from authors_ut
```

ユタ州に住む作家が authors テーブルに追加されたり削除されたりすると、authors_ut ビューは更新された authors テーブルを反映します。

ビューは、データベース内に物理的に格納されるデータを持つ 1 つ以上の実テーブルから抽出されます。ビューを抽出するテーブルは、ベース・テーブルまたは基本となるテーブルと呼ばれます。別のビューからビューを抽出することもできます。

ビューの定義は、ベース・テーブルからの抽出についてはデータベース内に格納されています。データの個々のコピーはこの格納された定義には関連付けられていません。ビューで参照するデータは、基本となるテーブルに格納されています。

ビューは、外見上では他のデータベース・テーブルとまったく同じです。他のテーブルとほぼ同様に、表示したり、作業を行うことができます。ビューを使用した問い合わせについては制限がなく、更新についても通常より制限が少なくなっています。これらの制限については、この章の後半で説明します。

ビューのデータを更新するときには、実際には基本となるベース・テーブルのデータを変更しています。逆に言えば、基本となるベース・テーブルのデータに行われた変更は、そのテーブルから抽出されたビューに自動的に反映されます。

ビューの利点

ユーザは、ビューを使用して、データベース内の知りたい情報に焦点をあて、単純化し、カスタマイズできます。また、使いやすいセキュリティの手段としての役割も果たします。さらに、データベースの構造に変更が加えられた場合に、ユーザが使いなれた方法でデータベースを作業するというときに便利です。

ビューは、次のように使用できます。

- 各ユーザに関係のあるデータ、およびそのユーザに責任があるタスクにフォーカスすることができます。必要のないデータは、ビューから除外できます。
- 頻繁に使用するジョイン、射影、選択をビューとして定義して、データに操作を実行するたびにすべての条件と修飾を指定する必要がなくなります。
- 同じデータを同時に使用する場合でも、ユーザ別に違うデータを表示できます。この機能はそれぞれ異なる作業を行う、さまざまなレベルの複数のユーザが同じデータベースを共有する場合に特に役立ちます。

セキュリティ

ビューを使用して表示できるデータに対してだけ、問い合わせや変更ができます。ビューに定義されていないデータベースの部分は、参照することも、アクセスすることもできません。

grant コマンドと revoke コマンドを使用すると、データベースへの各ユーザのアクセスを、ビューを含む、指定のデータベース・オブジェクトに制限できます。あるビューと、その抽出元となるすべてのテーブルとそれ以外のビューが同じユーザに所有されている場合、ユーザは、そのビューを使用するパーミッションを他のユーザに付与する一方、基本となるテーブルとビューを使用するパーミッションを拒否することができます。これは簡単ですが有効なセキュリティ・メカニズムです。『システム管理ガイド 第 1 巻』の「第 17 章 ユーザ・パーミッションの管理」を参照してください。

異なるビューを定義して、そのビューに対するパーミッションを選択して付与することによって、ユーザを異なるサブセットのデータに制限できます。たとえば、アクセスを次のように制限できます。

- アクセスをベース・テーブルのローのサブセット、つまり値に依存するサブセットに制限できます。たとえば、ビジネスと心理学の本のローだけを含むビューを定義して、その他のタイプの本についての情報を一部のユーザから見えないようにすることができます。
- アクセスをベース・テーブルのカラムのサブセット、つまり値に依存しないサブセットに制限できます。たとえば、titles テーブルの royalty カラムと advance カラムを除くすべてのローがあるビューを定義できます。
- アクセスをベース・テーブルのローとカラムのサブセットに制限できます。
- アクセスを、複数のベース・テーブルのジョインの条件を満たすローに制限できます。たとえば、作家とその作家の書いた本の名前を表示するために、titles、authors、titleauthors をジョインするビューを定義できます。ただし、このビューは、作家についての個人的な情報や、その本についての金銭的な情報は表示しません。
- アクセスをベース・テーブル内のデータの統計情報に制限できます。たとえば、category_price というビューを通して、ユーザは各タイプの本の平均価格だけにアクセスできます。
- アクセスを別のビューのサブセット、またはビューとベース・テーブルの組み合わせのサブセットに制限できます。たとえば、hiprice_computer というビューを通して、ユーザは hiprice のビュー定義にある条件を満たすコンピュータ関連の本のタイトルと価格にアクセスできます。

ビューを作成するには、データベース所有者によって create view パーミッションを付与される必要があります。また、ビュー定義内で参照されるテーブルやビューに対する適切なパーミッションを持っている必要があります。

ビューが異なるデータベース内でオブジェクトを参照している場合、ビューのユーザはそれぞれのデータベース内の有効なユーザまたは guest である必要があります。

他のユーザがビューを作成したオブジェクトを所有している場合は、どのユーザがどのビューを通してどのデータを参照できるのかを知っておく必要があります。たとえば、データベース所有者が “Harold” に create view パーミッションを与え、さらに “Maude” が、自分の所有するテーブルから select を実行するパーミッションを “Harold” に与えたとします。この場合、Harold は Maude が所有するテーブルからすべてのカラムとローを選択するビューを作成できます。Maude が所有するテーブルで select を実行するパーミッションを Harold から取り消した場合でも、Harold は自分が作成したビューを使用して Maude のデータを見ることができます。

論理データの独立性

ビューは、実テーブルの構造の変更が必要になった場合に、このような変更からユーザを保護します。

たとえば、select into を使用して titles テーブルを 2 つの新しいベース・テーブルに分割してから、titles テーブルを削除することによって、データベースを再編成するとします。

```
titletext (title_id, title, type, notes)
titlenumbers (title_id, pub_id, price, advance, royalty,
total_sales, pub_date)
```

元の titles テーブルは、2 つの新しいテーブルの title_id カラムをジョインすることによって、再生成できます。2 つの新しいテーブルのジョインのビューを作成できます。これに titles という名前を付けることができます。

ベース・テーブル titles を参照しているクエリやストアド・プロシージャは、ビュー titles を参照するようになりました。select オペレーションはこれまでとまったく同様に機能します。新しいビューからしか検索を行わないユーザは、再編成が行われたことを知る必要もありません。

今のところ、ビューは部分的な論理独立性しか提供しません。新しい titles でのデータ修正文のなかには、特定の制限のために許可されないものもあります。

例

まずは titles テーブルから抽出したビューの例を示します。価格が 15 ドルを超え、5000 ドルより多くの前払い金が支払われている本だけを抽出するとします。この簡単な select 文で、条件を満たすローを検索できます。

```
select *
from titles
where price > $15
  and advance > $5000
```

このデータ集合で、何度も検索と更新を行うとします。前述のクエリの条件を任意のコマンドと結合することや、関係のあるレコードのみが表示されるビューを作成することもできます。

```
create view hiprice
as select *
from titles
where price > $15
  and advance > $5000
```

Adaptive Server がこのコマンドを受け取ると、ビュー hiprice の定義である select 文をシステム・テーブル syscomments に格納します。また sysobjects および syscolumns にも、ビューの各カラムが入力されます。

ここで hiprice を表示したり操作したりすると、格納されている hiprice の定義とユーザの文が結合されます。たとえば、hiprice にあるすべての価格を、他のテーブルで変更を行うのと同様にして変更することができます。

```
update hiprice
set price = price * 2
```

Adaptive Server はシステム・テーブル内でビュー定義を検出し、この update コマンドを次の文に変換します。

```
update titles
set price = price * 2
where price > $15
  and advance > $5000
```

言い換えると、Adaptive Server は更新されるデータが titles にあることを、ビュー定義から判断します。また、ビュー定義で指定された price カラムと advance カラムの条件、および update 文の条件を満たすローでだけ価格を引き上げます。

hiprice の update 文を発行すると、ユーザはビューと titles テーブルのどちらででもその影響を確認できます。逆に、ビューを作成してから、ベース・テーブルを直接操作する次の update 文を発行した場合、変更された価格はビューを通して参照できます。

異なるローがビューの条件を満たすこのような方法でビューの基本となるテーブルを更新すると、ビューに影響します。たとえば、『You Can Combat Computer Stress』という本の価格を 25.95 ドルに上げるとします。この本は、この時点でビュー定義文の条件を満たすので、ビューの一部と考えられます。

しかし、カラムを追加することによってビューの基本となるテーブルの構造を変更した場合は、ビューを削除して再定義しないと、select * 句で定義されたビューに新しいカラムは表示されません。これは、元のビュー定義のアスタリスクが元のカラムだけを対象としているためです。

ビューの作成

ビュー名は既存のテーブルおよびビューの間でユーザごとにユニークである必要があります。set quoted_identifier on を設定した場合、ビューに区切り識別子を使用できます。設定していない場合は、「識別子」(10 ページ) に説明されている識別子の規則に従ってください。

他のビューや、ビューを参照するプロシージャに、ビューを構築することができます。ビューには、プライマリ・キー、外部キー、共通キーを定義できます。ただし、ビューにルール、デフォルト、またはトリガを関連付けたり、ビューにインデックスを構築することはできません。一時的なビューを作成したり、テンポラリ・テーブルにビューを作成することはできません。

create view 構文

「例」(392 ページ) の create view の例に示されるように、ビュー定義文の create 句にカラム名を指定する必要はありません。Adaptive Server は、select 文の select リスト内で参照されるカラムと同じ名前とデータ型を持つビューのカラムを提供します。select リストは、例に示されるようにアスタリスク (*) で指定したり、ベース・テーブルのカラム名を完全にまたは部分的にリストすることもできます。

『リファレンス・マニュアル：コマンド』を参照してください。

ローが重複しないビューを構築するには、select 文の distinct キーワードを使用して、ビューの各ローがユニークになるようにします。しかし、distinct ビューは更新できません。

ビューのカラムの名前を、基本となるテーブルにある名前とは異なる名前にするビュー定義文を次に示します。

```
create view pub_view1 (Publisher, City, State)
as select pub_name, city, state
from publishers
```

同じビューを作成し、select 文でカラムの名前を変更するという方法を次に示します。

```
create view pub_view2
as select Publisher = pub_name,
City = city, State = state
from publishers
```

次の項に示すビュー定義文の例は、create 句にカラム名を含めるためのその他の規則です。

注意 ローカル変数は、ビュー定義に使用できません。

create view での *select* 文の使用

create view 文内の select 文はビューを定義します。ユーザは、作成しているビューの select 文内で参照されるオブジェクトから select を実行するパーミッションが必要です。

複雑な select 文を使って、複数のテーブルおよび他のビューを使用するビューを作成できます。

ビュー定義の select 文には、いくつかの制限があります。

- order by または compute 句を含めることはできません。
- into キーワードを含めることはできません。
- テンポラリ・テーブルを参照することはできません。

ビューを作成すると、syscomments システム・テーブルの text カラムに、ビューを記述する「ソース・テキスト」が格納されます。

注意 syscomments からこの情報を削除しないでください。代わりに、sp_hidetext を使用して、syscomments 内でテキストを暗号化します。『リファレンス・マニュアル：プロシージャ』および「コンパイル済みオブジェクト」(3 ページ)を参照してください。

射影を使用したビュー定義

titles テーブルのすべてのローを含むが、テーブルのカラムのサブセットを 1 つしか含まないビューを作成するには、次の文を入力します。

```
create view titles_view
as select title, type, price, pubdate
from titles
```

create view 句にカラム名は含まれていません。ビュー titles_view は、select リストに指定されたカラム名を継承します。

計算カラムを使用したビュー定義

カラム price、royalty、total_sales から生成された計算カラムを使用したビューを作成するビュー定義文を次に示します。

```
create view accounts (title, advance, amt_due)
as select titles.title_id, advance,
(price * royalty /100) * total_sales
from titles, roysched
where price > $15
and advance > $5000
and titles.title_id = roysched.title_id
and total_sales between lorange and hirange
```

price、royalty、total_sales を掛け合わせて計算したカラムに、継承できる名前がないので、create 句にカラムのリストを含めます。計算されたカラムは amt_due と名付けられます。このカラムは、このカラムを計算した式が select 句内でリストされているのと同じ位置に、create 句内でリストする必要があります。

集合関数または組み込み関数を使用したビュー定義

集合関数または組み込み関数を含んでいるビュー定義には、create 句内にカラム名を含める必要があります。次に例を示します。

```
create view categories1 (category, average_price)
as select type, avg(price)
from titles
```

```
group by type
```

セキュリティ管理のためにビューを作成する場合は、集合関数と group by 句を使用するときに注意が必要です。group by を伴う select に含めることができるカラムの制限がない Transact-SQL 拡張機能では、ビューが必要以上の情報を返すことがあります。次に例を示します。

```
create view categories2 (category, average_price)
as select type, avg(price)
from titles
where type = "business"
```

ビューの結果を “business” カテゴリに制限しようとしたのですが、結果にはその他のカテゴリの情報が含まれます。「クエリ結果のグループ構成：group by 句」(75 ページ) を参照してください。

ジョインを使用したビュー定義

複数のベース・テーブルから抽出したビューを作成できます。authors テーブルと publishers テーブルの両方から抽出したビューの例を次に示します。ビューには、出版社の名前および都市とともに、出版社と同じ都市に住む作家の名前と都市が含まれます。

```
create view cities (authorname, acity, publishername, pcity)
as select au_lname, authors.city, pub_name, publishers.city
from authors, publishers
where authors.city = publishers.city
```

外部ジョインで使用されるビュー

外部ジョインでビューを定義してから、外部ジョインの内部テーブルからのカラムに対する修飾を使用してそのビューに問い合わせた場合、クエリは修飾がビューの外部ジョインの on 句の一部ではなくビューの where 句の一部であるかのように動作します。したがって、修飾は、外部ジョインの完了後にローのみに対して作用します。たとえば、外部ジョイン条件が満たされた場合は null で拡張されたローに対して修飾が作用し、それに従ってローを除外します。

次の規則は、ジョイン・ビューからカラムに対して実行できる更新の種類を決定します。

- delete 文はジョイン・ビューでは許可されません。
- insert 文は with check option で作成されたジョイン・ビューには許可されません。
- update 文は with check option のジョイン・ビューで許可されます。影響を受けるカラムが、複数のテーブルからのカラムを含む式の where 句に表示される場合、更新は失敗します。
- ジョイン・ビューからローを挿入または更新した場合、影響を受けるすべてのカラムは同一のベース・テーブルに属する必要があります。

他のビューから抽出したビュー

次の例に示すように、あるビューを別のビューを使って定義できます。

```
create view hiprice_computer
as select title, price
from hiprice
where type = "popular_comp"
```

distinct ビュー

次の例に示すように、あるビューに含まれるローをユニークにすることができます。

```
create view author_codes
as select distinct au_id
from titleauthor
```

ローのカラム値が、別のローに含まれるカラム値とすべて一致する場合、そのローは複製です。2 つの null 値は同一とされます。

Adaptive Server は初めてビューにアクセスするときに、distinct 要件をビューの定義に適用してから、射影または選択を行います。ビューの概観および機能は、他のデータベース・テーブルとまったく同じです。distinct ビューの射影を選択する (つまり、ビューのカラムをいくつかだけ選択し、そのローをすべて選択する) と、複製のような結果が得られます。しかし、ビューそのものの各ローはユニークです。たとえば、次に示す値を含む a、b、c という 3 つのカラムを持つ myview という distinct ビューを作成するとします。

a	b	c
1	1	2
1	2	3
1	1	0

次のクエリを入力した場合、

```
select a, b from myview
```

結果は次のようになります。

```
a    b
---  ---
1    1
1    2
1    1

(3 rows affected)
```

最初のローと 3 つ目のローは重複しているように見えます。しかし、基本となるビューのローはユニークです。

IDENTITY カラムを含むビュー

カラム名、つまり syb_identity キーワードをビューの select 文にリストすることによって、IDENTITY カラムを含むビューを定義できます。次に例を示します。

```
create view sales_view
as select syb_identity, stor_id
from sales_daily
```

ただし、*identity_column_name* = identity(*precision*) 構文を使用して新しい IDENTITY カラムをビューに追加することはできません。

ビューに次の事柄が当てはまらないかぎり、syb_identity キーワードを使用して IDENTITY カラムをビューから選択することができます。

- IDENTITY カラムを複数回選択する
- IDENTITY カラムから新しいカラムを計算する
- 集合関数を含む
- 複数のテーブルからカラムをジョインする
- IDENTITY カラムを式の一部として含む

以上の条件のいずれかに当てはまる場合、そのビューに関しては、Adaptive Server はカラムを IDENTITY カラムとは認識しません。ビューで sp_help を実行すると、カラムは “Identity” 値 0 を表示します。

次の例では、row_id カラムは store_discounts ビューに関しては IDENTITY カラムとして認識されません。store_discounts は 2 つのテーブルからカラムをジョインしているためです。

```
create view store_discounts
as
select stor_name, discount
from stores, new_discounts
where stores.stor_id = new_discounts.stor_id
```

ビューを定義するときに、基本となるカラムは IDENTITY プロパティを保持します。ビューを通してローを更新するときは、IDENTITY カラムに新しい値を指定できません。ビューを通してローを挿入すると、Adaptive Server は IDENTITY カラムに新しい連続する値を生成します。IDENTITY カラムのベース・テーブルに identity_insert on を設定した後で IDENTITY カラムに値を明示的に挿入できるのは、テーブル所有者、データベース所有者、またはシステム管理者だけです。

ビューの選択基準の検証

通常は、影響を受けるローがビューのスコープ内にあるかどうかを判断するために、Adaptive Server がビューの insert および update 文をチェックすることはありません。ビューの選択基準を満たさなくなるようにビューに挿入するのではなく基本となるベース・テーブルにローを挿入したり既存のローを変更したりすることができます。

with check option 句を使用してビューを作成すると、ビューを通した insert および update は、それぞれビューの選択基準を満たすかどうか検証されます。ビューを通して挿入または更新されたローは、すべてビューを通して参照できるようになっている必要があります。そうなっていない場合、文は失敗します。

with check option を使用して作成したビュー、stores_ca の例を次に示します。このビューには、カリフォルニア州にある支店の情報が含まれていますが、他の州にある支店の情報は除外されています。ビューは、state の値が “CA” であるすべてのローを stores テーブルから選択することによって作成されています。

```
create view stores_ca
as select * from stores
where state = "CA"
with check option
```

stores_ca を通してローを挿入しようとすると、Adaptive Server は新しいローがビューのスコープ内にあるかどうかを検証します。次の insert 文は、新しいローの state の値が “CA” ではなく “NY” であるため、失敗します。

```
insert stores_ca
values ("7100", "Castle Books", "351 West 24 St.", "New York",
"NY", "USA", "10011", "Net 30")
```

stores_cal を通してローを更新しようとすると、Adaptive Server は更新によってビューからローが削除されないかどうかを検証します。次の update 文は、state の値を “CA” から “MA” に変更するため失敗します。この更新が実行されると、ビューを使用したローの参照ができなくなります。

```
update stores_ca
set state = "MA"
where stor_id = "7066"
```

他のビューから抽出したビュー

with check option を使用してビューを作成すると、「ベース」ビューから抽出されたビューはすべてそのチェック・オプションの条件を満たしていなければなりません。抽出されたビューを通して挿入されたローは、ベース・ビューを通して参照できる必要があります。抽出されたビューを通して更新されたローは、ベース・ビューを通して参照できる必要があります。

stores_cal から抽出されたビュー stores_cal30 を考えてみます。新しいビューには、“Net 30” という支払期限が指定されているカリフォルニア州にある支店の情報が含まれます。

```
create view stores_cal30
as select * from stores_ca
where payterms = "Net 30"
```

stores_cal は with check option を使用して作成されているため、stores_cal30 を通して挿入または更新されたローは、すべて stores_cal を通して参照できる必要があります。“CA” 以外の state の値を持つローは拒否されます。

stores_cal30 には、それ自身の with check option 句がないことに注意してください。これは、“Net 30” 以外の payterms 値を持つローを、stores_cal30 を介して挿入または更新できることを示します。次の update 文は、stores_cal30 を通してローを参照することはできなくなりますが、正常に実行されます。

```
update stores_cal30
set payterms = "Net 60"
where stor_id = "7067"
```

ビューを通したデータ検索

ビューを通してデータを検索するときに、Adaptive Server は、文中で参照されるデータベース・オブジェクトがすべて存在し、文のコンテキストにおいて有効であることを検証します。確認できると、Adaptive Server はビューの格納されている定義と文を結合して、ビューの基本となるテーブルでのクエリに変換します。このプロセスは「ビューの解析」と呼ばれます。

次のビュー定義文、およびそれに対するクエリを考えてみます。

```
create view hiprice
as select *
from titles
where price > $15
and advance > $5000
select title, type
from hiprice
where type = "popular_comp"
```

Adaptive Server は、内部的に hiprice のクエリをその定義と結合し、次のように変換します。

```
select title, type
from titles
where price > $15
and advance > $5000
and type = "popular_comp"
```

一般的に、実テーブルであるかのように、任意のビューを任意の方法で問い合わせることができます。ジョイン、group by 句、サブクエリ、その他のクエリを、任意に組み合わせてビューに使用できます。ただし、ビューが外部ジョインまたは集合関数を使用して定義されている場合は、ビューに問い合わせると予期しない結果が返される場合があります。「他のビューから抽出したビュー」(397 ページ) を参照してください。

注意 ビューの text および image カラムに、select を使用できます。しかし、ビューで readtext と writetext を使用することはできません。

ビューの解析

ビューを定義するときに、Adaptive Server は、from 句にリストされているすべてのテーブルまたはビューが存在するか確認します。ビューを通して問い合わせるときも同様の確認が行われます。

ビューが定義された時点と、文でビューが使用される時点の間に、事情が変わっている場合があります。たとえば、ビュー定義の from 句にリストされていた 1 つ以上のテーブルまたはビューが削除されている場合があります。または、ビュー定義の select 句にリストされる 1 つ以上のカラムの名前が変更されている場合があります。

ビューを完全に解析するために、Adaptive Server は次のことを確認します。

- ビューが抽出されたすべてのテーブル、ビュー、カラムが存在する。
- ビューのカラムが依存する各カラムのデータ型が、互換性のない型に変更されていない。
- 文が update、insert、delete の場合は、その文がビューの修正の制限を違反していない。詳細については、「ビューを通したデータ修正」(403 ページ) を参照。

以上のいずれかが確認できないと、Adaptive Server はエラー・メッセージを返します。

ビューの再定義

再定義によって Adaptive Server が従属ビューを変換できなくなる場合を除いて、ビューに従属しているその他のビューを再定義しなくてもビューを再定義できます。

例として、authors テーブルとそこから作成できる 3 つのビューを次に示します。view2 は view1 から、view3 は view2 からというように、後続のビューはそれぞれ前のビューを使用して定義されます。このように、view2 は view1 に従属し、view3 はその前の 2 つのビューに従属します。

それぞれのビュー名の後に、その文の作成に使用した select 文を示します。

view1：

```
create view view1
as select au_lname, phone
from authors
where postalcode like "94%"
```

view2：

```
create view view2
as select au_lname, phone
from view1
where au_lname like "[M-Z]%"
```

view3：

```
create view view3
as select au_lname, phone
from view2
where au_lname = "MacFeather"
```

これらのビューの基本となる authors テーブルは、au_id、au_lname、au_fname、phone、address、city、state、postalcode カラムで構成されています。

view2 を削除して、次のような少しだけ異なる選択基準を含む view2 という同じ名前の別のビューに置き換えることができます。

```
create view view2
as select au_lname, phone
from view3
where au_lname like "[M-P]"
```

view2 に従属する view3 はこの時点でも有効であり、再定義の必要はありません。view2 または view3 のいずれかを参照するクエリを使用すると、通常どおりにビューの解析が行われます。

view3 が抽出されないように view2 を定義した場合、view3 は無効になります。たとえば、view2 の新しい別のバージョンに、view3 が期待する 2 つのカラムではなく au_lname という 1 つのカラムが含まれている場合、従属するオブジェクトから phone カラムを抽出できないため、view3 は使用できません。

ただし view3 はこの時点でも存在しています。view2 を削除して au_lname カラムと phone カラムの両方を指定した view2 を再作成することによって、view3 を再度使用できます。

つまり、従属ビューの select リストが有効であるかぎり、従属ビューに影響することなく、中間ビューの定義を変更することができます。この規則に違反すると、無効なビューを参照するクエリはエラー・メッセージを生成します。

ビュー名の変更

次を使用するとビューの名前を変更できます。

sp_rename *objname* , *newname*

たとえば、titleview を bookview に変更するには、次のように入力します。

```
sp_rename titleview, bookview
```

ビュー名を変更するときは、次の規則に従います。

- 新しい名前が「識別子」(10 ページ) に説明されている識別子の規則に従っていることを確認してください。
- 変更できるのは、自分が所有するビューの名前だけです。データベース所有者は、どのユーザが所有するビューの名前も変更できます。
- ビューが現在のデータベースにあることを確認してください。

基本となるオブジェクトの変更または削除

ビューの基本となるオブジェクトの名前を変更できます。たとえば、ビューが new_sales というテーブルを参照している場合、そのテーブルの名前を old_sales に変更すると、ビューは新しい名前のテーブルで機能します。

ただし、ビューが参照しているテーブルが削除された場合、そのビューを使用しようとすると、Adaptive Server はエラー・メッセージを生成します。削除されたテーブルまたはビューに代わる新しいテーブルまたはビューが作成されると、ビューは再度使用可能になります。

select * 句を使用してビューを定義した場合、カラムを追加してビューの基本となるテーブルの構造を変更しても、新しいカラムは表示されません。これは、省略形のアスタリスクが、ビューの作成時に解釈され、拡張されるためです。新しいカラムを参照するには、ビューを削除して再作成してください。

ビューを通したデータ修正

Adaptive Server ではビューを通したデータの検索に制限がなく、Transact-SQL ではビューを通したデータの修正に他のバージョンの SQL ほど多くの制限がありませんが、さまざまなデータ修正オペレーションには次の規則が適用されます。

- ビューの計算カラムや組み込み関数を参照する update、insert、または delete オペレーションは許可されていません。
- 集合関数やロー集合を含むビューを参照する update、insert、または delete オペレーションは許可されていません。

- distinct ビューを参照する insert、delete、update オペレーションは許可されていません。
- 新しいローの挿入を行うビューの基本となるテーブルまたはビューのすべてに not null カラムが含まれていないかぎり、insert 文は許可されていません。Adaptive Server は、基本となるオブジェクトの not null カラムに値を提供できません。
- ビューが with check option 句を使用している場合、そのビュー (またはその抽出ビュー) を通して挿入または更新されたローは、すべてビューの選択基準を満たす必要があります。
- 複数のテーブルから構成されるビューでの delete 文は許可されていません。
- with check option 句を使用して作成された複数のテーブルから構成されるビューでの insert 文は許可されていません。
- with check option 句が使用されている複数のテーブルから構成されるビューでは update 文が許可されています。影響を受けるカラムが、複数のテーブルからのカラムを含む式の where 句に表示される場合、更新は失敗します。
- 複数のテーブルから構成される distinct ビューでの insert および update 文は許可されていません。
- update 文は IDENTITY カラムに値を指定できません。テーブル所有者、データベース所有者、またはシステム管理者は、IDENTITY カラムのベース・テーブルに identity_insert on を設定した後で IDENTITY カラムに明示的に値を insert できます。
- 複数のテーブルから構成されるビューを通してローを挿入または更新する場合、影響を受けるカラムはすべて同じベース・テーブルに属す必要があります。
- ビューの text カラムおよび image カラムでの writetext は許可されていません。

ビューに update、insert、または delete を実行しようとすると、Adaptive Server は、以上の制限に違反していないか、およびデータ整合性の規則に違反していないかを確認します。

ビューの更新の制限

更新されるビューの制限は、次の領域に適用されます。

- ビュー定義内の計算カラム
- ビュー定義の group by または compute
- 基本となるオブジェクトの null 値
- with check option を使用して作成されたビュー

- 複数のテーブルから構成されるビュー
- IDENTITY カラムを使用したビュー

ビュー定義内の計算カラム

この制限は、計算カラムまたは組み込み関数から抽出されるビューのカラムに適用されます。たとえば、ビュー accounts の amt_due カラムは計算カラムです。

```
create view accounts (title_id, advance, amt_due)
as select titles.title_id, advance,
(price * royalty /100) * total_sales
from titles, roysched
where price > $15
  and advance > $5000
and titles.title_id = roysched.title_id
and total_sales between lorange and hirange
```

accounts を通して参照できるローは次のとおりです。

```
select * from accounts

title_id     advance     amt_due
--------     --------    ---------
PC1035      7,000.00     32,240.16
PC8888      8,000.00      8,190.00
PS1372      7,000.00        809.63
TC3218      7,000.00        785.63

(4 rows affected)
```

amt_due カラムへの updates および inserts は許可されていません。これは、amt_due カラムに入力する値からは、価格、印税、または現在までの売上の基本となる値を推測できないためです。delete オペレーションは、削除する基本となる値がないので意味がありません。

ビュー定義の *group by* または *compute*

この制限は、集約値を含むビュー、つまり定義に group by または compute 句が含まれるビューのすべてのカラムに適用されます。group by 句を使用したビュー定義と、そのビューを通して参照されるローを次に示します。

```
create view categories (category, average_price)
as select type, avg(price)
from titles
group by type


select * from categories

category         average_price
-------------    -------------
UNDECIDED                NULL
```

```
business                13.73
mod_cook                11.49
popular_comp            21.48
psychology              13.50
trad_cook               15.96

(6 rows affected)
```

ビューの categories へのローの insert はできません。これは、挿入されたローが属するグループを特定できないためです。average_price カラムでの更新は許可されていません。これは、このカラムに入力する値からは、基本となる価格がどのように変更されるか判断できないからです。

基本となるオブジェクトの null 値

この制限は、ビューが抽出されるテーブルまたはビューにいくつかの not null カラムが含まれるときに、insert 文に適用されます。

たとえば、ビューの基本となるテーブルのカラムで、null 値が許可されていないとします。通常は、ビューを通して新しいローの insert を実行すると、基本となるテーブルのビューに含まれていないカラムには、null 値が指定されます。これらのカラムの 1 つ以上で null 値が許可されていない場合、ビューを通して挿入することはできません。

たとえば、このビューでは次のようになります。

```
create view business_titles
as select title_id, price, total_sales
from titles
where type = "business"
```

基本となるテーブル titles の title カラムでは null 値は許可されていないので、business_view を通して insert 文を実行することはできません。title カラムがビューに存在しなくても、そのカラムへの null が許可されていないことによって、ビューへの挿入は無効になります。

同様に、title_id カラムにユニーク・インデックスが含まれる場合、ビューの値には重複しなくても、基本となるテーブルの値と重複する更新または挿入は拒否されます。

with check option を使用して作成されたビュー

この制限は、with check option を使用したビューを通して実行できる変更のタイプを決定します。ビューが with check option 句を使用している場合、そのビューを通して挿入または更新された各ローは、ビュー内で参照できる必要があります。これは、別の抽出ビューを通して直接または間接的にビューを挿入、更新する場合にも当てはまります。

複数のテーブルから構成されるビュー

この制限は、複数のテーブルからカラムをジョインするビューを通して実行できる変更のタイプを決定します。Adaptive Server は、複数のテーブルから構成されるビューでの delete 文を禁止していますが、update および insert 文は許可しています。これは他のシステムでは許可されていません。

次の条件に当てはまる場合は、複数のテーブルから構成されるビューに対して insert または update を実行できます。

- ビューが with check option 句を使用していない。
- 挿入または更新されるすべてのカラムが、同じベース・テーブルに属する。

たとえば、次に示すような、titles と publishers の両方からのカラムを含み、with check option 句を使用していないビューを考えてみます。

```
create view multitable_view
as select title, type, titles.pub_id, state
from titles, publishers
where titles.pub_id = publishers.pub_id
```

1 つの insert または update 文は、titles からのカラムか、publishers からのカラムの、どちらかの値を指定できます。

```
update multitable_view
set type = "user_friendly"
where type = "popular_comp"
```

しかしこの文は、titles と publishers の両方のカラムに影響するので、失敗します。

```
update multitable_view
set type = "cooking_trad",
state = "WA"
where type = "trad_cook"
```

IDENTITY カラムを使用したビュー

この制限は、IDENTITY カラムを含むビューに対して実行できる修正のタイプを決定します。定義により、IDENTITY カラムは更新できません。ビューを通した更新は、IDENTITY カラムの値を指定できません。

次のユーザしか IDENTITY カラムへの挿入はできません。

- テーブル所有者
- テーブル所有者によってパーミッションを付与されている、データベース所有者またはシステム管理者
- setuser コマンドでテーブル所有者になれる、データベース所有者またはシステム管理者

ビュー経由のこのような挿入を可能にするには、カラムのベース・テーブルに対して set identity_insert on を実行します。挿入を実行するビューに対して set identity_insert on は使用できません。

ビューの削除

データベースからビューを削除するには、次を使用します。

drop view [*owner*.]*view_name* [, [*owner*.]*view_name*]...

前述のように、複数のビューを同時に削除できます。ビューを削除できるのは、その所有者 (またはデータベース所有者) だけです。

drop view コマンドを実行すると、ビューの情報が sysprocedures、sysobjects、syscolumns、syscomments、sysprotects、sysdepends から削除されます。そのビューについての権限も削除されます。

ビューが、削除されたテーブルまたは別のビューに従属する場合、そのビューを使用しようとすると、Adaptive Server はエラー・メッセージを返します。新しいテーブルまたはビューが作成されて削除されたものに置き換わり、削除されたテーブルまたはビューと同じ名前を持っている場合は、ビュー定義内で参照されるカラムが存在すれば、ビューは再度使用可能になります。

セキュリティ・メカニズムとしてのビューの使用

ビュー内のデータのサブセットにアクセスするためのパーミッションは、ビューの基本となるテーブルについて付与されているパーミッションとは関係なく、明示的に付与したり取り消したりする必要があります。ビューへのアクセスが許可されていても、その基本となるテーブルへのアクセスが許可されていないユーザは、基本となるテーブルのうち、ビューに含まれていないデータを参照することはできません。

たとえば、一部のユーザを、titles テーブル内の金銭と売上に関するカラムにアクセスできないようにするとします。金銭と売上に関するカラムを除いて titles テーブルのビューを作成し、そのビューに対するパーミッションをすべてのユーザに付与して、テーブルに対するパーミッションは営業部門にだけ付与するようにします。次に例を示します。

```
revoke all on titles to public
grant all on bookview to public
grant all on titles to sales
```

『システム管理ガイド 第 1 巻』の「第 17 章 ユーザ・パーミッションの管理」を参照してください。

ビュー情報の取得

システム・プロシージャ、カタログ・ストアド・プロシージャ、Adaptive Server 組み込み関数は、システム・テーブルからビューについての情報を提供します。『リファレンス・マニュアル：プロシージャ』を参照してください。

sp_help および *sp_helptext* を使用したビュー情報の表示

ビューのレポートを取得するには、sp_help を使用します。

```
sp_help hiprice
----------------
```

システム・セキュリティ担当者は、評価済み設定で allow select on syscomments.text column 設定パラメータをリセットする必要があります (詳細については、『用語解説』の「評価済み設定」を参照してください)。この場合、sp_helptext を通してビューのテキストを参照するには、ユーザはビューの作成者であるか、システム管理者である必要があります。

create view 文のテキストを表示するには、sp_helptext を実行します。

```
sp_helptext hiprice
# Lines of Text
-----------------
3

(1 row affected)

 text

 ------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------

--Adaptive Server has expanded all '*' elements in the following statement

 create view hiprice as select

titles.title_id, titles.title, titles.type, titles.pub_id, titles.price,
titles.advance, titles.total_sales, titles.notes, titles.pubdate,
titles.contract from titles where price > $15 and advance > $5000

(3 rows affected)
(return status = 0)
```

ビューのソース・テキストが sp_hidetext を使用して暗号化されている場合、Adaptive Server はテキストが隠されていることを通知するメッセージを表示します。『リファレンス・マニュアル：プロシージャ』を参照してください。

sp_depends を使用した従属オブジェクトのリスト

sp_depends は、現在のデータベースでビューやテーブルが参照するすべてのオブジェクトをリストし、そのビューまたはテーブルを参照するすべてのオブジェクトをリストします。

```
sp_depends titles

Things inside the current database that reference the object.
object                type
------------          --------------------------
dbo.history_proc      stored procedure
dbo.title_proc        stored procedure
dbo.titleid_proc      stored procedure
dbo.deltitle          trigger
dbo.totalsales_trig   trigger
dbo.accounts          view
dbo.bookview          view
dbo.categories        view
dbo.hiprice           view
dbo.multitable_view   view
dbo.titleview         view

(return status = 0)
```

データベース内のすべてのビューのリスト

sp_tables を次のフォーマットで使用すると、データベース内のすべてのビューがリストされます。

```
sp_tables @table_type = "'VIEW'"
```

オブジェクト名および ID の表示

システム関数 object_id および object_name は、ビューの ID と名前を特定します。次に例を示します。

```
select object_id("titleview")

----------
 480004741
```

オブジェクト名および ID は、sysobjects テーブルに格納されます。

第 13 章

テーブルのインデックスの作成

テーブルにある指定したカラムの値に基づいて「インデックス」を作成すると、テーブルにあるデータにすばやくアクセスできます。テーブルには、複数のインデックスを作成できます。インデックスはそのテーブルからのデータをアクセスするユーザに対して透過的で、テーブルに作成されたインデックスをいつ使用するかは、Adaptive Server が自動的に決定します。

パフォーマンスを上げるためのインデックスの設計方法については、『パフォーマンス&チューニング・シリーズ：ロックと同時実行制御』を参照してください。分割されたインデックスの作成と管理については、「第 10 章 テーブルとインデックスの分割」を参照してください。

インデックスの機能

インデックスはディスク上にあるテーブル・カラムのデータの位置を指すため、データの検索が速くなります。たとえば、stores テーブルに保管された ID 番号を使用して、頻繁にクエリを実行する必要があるとします。Adaptive Server は stores テーブルの各ローを 1 つ 1 つ検索するため、数百万のローが含まれている場合はかなり時間がかかることがあります。このため、テーブルの各ローを検索しないように、次に示す stor_id_ind というインデックスを作成できます。

```
create index stor_id_ind
on stores (stor_id)
```

stor_id_ind インデックスは、次回 stores の stor_id カラムを問い合わせたときに自動的に有効になります。つまり、インデックスはユーザにとって透過的です。SQL には、クエリでインデックスを参照するための構文はありません。テーブルからインデックスを作成または削除することだけが可能で、各クエリがそのテーブルに実行されたときにインデックスを使用するかどうかは Adaptive Server が決定します。後でテーブルのデータが変更されると、Adaptive Server はそれらの変更を反映させるため、テーブルのインデックスを変更します。これらの変更も、ユーザにとっては透過的です。

Adaptive Server は、次の種類のインデックスをサポートします。

- 「複合インデックス」– このインデックスは、2 つ以上のカラムを利用します。複数のカラムに論理的な関係があり、1 つの単位として検索することが最良である場合に、このタイプのインデックスを使用します。
- 「ユニーク・インデックス」– このインデックスは、指定したカラム内の 2 つのローに同じ値を許可しません。既にデータが存在する場合は、インデックスの作成時、またはデータを追加するたびに、Adaptive Server では重複値がないかどうかをチェックします。
- 「クラスタード・インデックス」または「ノンクラスタード・インデックス」– クラスタード・インデックスを使用すると、Adaptive Server は強制的にソートおよび再ソートを連続して実行し、テーブルのローの物理的な順序を常に論理的な順序 (またはインデックスの順序) に一致させるようにします。クラスタード・インデックスは、1 つのテーブルに対して 1 つだけ作成できます。ノンクラスタード・インデックスは、ローの物理的な順序をインデックスの順序に一致させる必要はありません。各ノンクラスタード・インデックスは、異なるソート順でデータをアクセスできます。
- 「ローカル・インデックス」– ローカル・インデックスは、1 つのデータ・パーティションのみにインデックス付けを行うインデックスのサブツリーです。ローカル・インデックスは分割可能で、すべての種類の分割されたテーブルでサポートされています。
- 「グローバル・インデックス」– テーブルのすべてのデータ・パーティションにインデックス付けを行います。分割されていないグローバル・クラスタード・インデックスは、ラウンドロビン方式で分割されたテーブルでサポートされ、ノンクラスタード・グローバル・インデックスは、すべての種類の分割されたテーブルでサポートされています。グローバル・インデックスは分割できません。

ローカル・インデックスとグローバル・インデックスについては、「第 10 章 テーブルとインデックスの分割」を参照してください。その他の種類のインデックスについては、この章でより詳しく説明します。

インデックスを作成する 2 つの方法の比較

テーブルにインデックスを作成するには、create index 文 (この章で説明) を使用する方法と、create table コマンドのunique または primary key 整合性制約を使用する方法の 2 つがあります。しかし、整合性制約には、次のような制限があります。

- ユニークでないインデックスは作成できない。
- create index コマンドが提供するオプションを使用して、インデックスの動作を変更することはできない。
- これらのインデックスを削除できるのは、alter table 文を使用して制約として削除する場合だけである。

使用しているアプリケーションでこれらの機能が必要な場合は、create index を使用してインデックスを作成します。それ以外の場合は、unique または primary key 整合性制約を使用する方が、テーブルのインデックスをより簡単に定義できます。unique または primary key 制約の詳細については、「第 8 章 データベースおよびテーブルの作成」を参照してください。

インデックスの使用におけるガイドライン

カラムにインデックスを作成すると、クエリに対する応答に非常に差が出てくることがあります。

ただし、インデックスの構築には時間と記憶領域が必要です。たとえば、クラスタード・インデックスを再構築するときに、ノンクラスタード・インデックスも自動的に再作成されます。

また、インデックスの作成されたカラムへのデータの挿入、削除、更新は、インデックスのないカラムより時間がかかります。しかし、通常これらにかかる時間は、インデックスによる検索のパフォーマンスの向上を考えれば、それほど問題ではありません。

インデックスを作成すべきかどうかを判断するときは、次の一般的ガイドラインに従ってください。

- IDENTITY カラムに手動でデータを挿入する場合は、カラムで既に使用されている値を挿入できないように、ユニーク・インデックスを作成する。
- order by で指定されたソート順で頻繁にアクセスされるカラムには一般に、インデックスを作成すべきである。これによって、Adaptive Server はインデックスの順序を利用できるようになる。
- 定期的にジョインで使用されるカラムには、常にインデックスを作成すべきである。カラム内の順序がソートされていると、システムはジョインをより高速に実行できる。

- テーブルのプライマリ・キーを保管するカラムには、常にクラスタード・インデックスを作成する。特に、これらのカラムが他のテーブルのカラムに頻繁にジョインされる場合はインデックスを作成する。クラスタード・インデックスは、1 つのテーブルに対して 1 つだけ作成できることに注意。
- 値の範囲が頻繁に検索されるカラムには、クラスタード・インデックスを選択するとよいことが多い。範囲内で最初の値を持つローが検出されると、続く値を持つローは物理的に隣接していることが保証される。クラスタード・インデックスは、1 つの値を検索する場合はあまり効果がない。

次に、インデックスが必要ない場合について説明します。

- クエリで参照されることがほとんど、またはまったくないカラムにインデックスを作成しても効果がない。システムは、これらのカラムの値を基にしてローを検索することがほとんどないからである。
- 多数の重複値を持ち、テーブルのロー数に比較してユニークな値をほとんど持たないカラムにインデックスを付けても実際的な効果はない。

システムは、インデックスの付いていないカラムを検索する必要がある場合、ローを 1 つずつ確認しながら検索します。このような検索では、検索にかかる時間の長さはテーブルのローの数に比例します。

インデックスの作成

create index を実行する前に、select into をオンにします。

```
sp_dboption,'select into', true
```

create index の最も簡単な形式は、次のとおりです。

```
create index index_name
    ontable_name (column_name)
```

authors テーブルの au_id カラムにインデックスを作成するには、次のコマンドを実行します。

```
create index au_id_ind
on authors(au_id)
```

インデックス名は、識別子の規則に従います。カラム名とテーブル名には、インデックスを付けるカラムとそれが含まれるテーブルを指定します。

bit、text、image データ型のカラムには、インデックスを作成することはできません。

インデックスを create または drop するには、テーブルの所有者でなければなりません。テーブルの所有者は、そのテーブルにデータが存在するかどうかにかかわらず、いつでもインデックスを create または drop できます。テーブル名を修飾することによって、別のデータベースにあるテーブルにインデックスを作成できます。

create index 構文

以降の項では、create index コマンドの各種のオプションについて説明します。index_partition_clause の使用法を含めたインデックス・パーティションの作成と管理については、「第 10 章 テーブルとインデックスの分割」を参照してください。

注意　create index の拡張構文である on *segment_name* を使用して、セグメント内にインデックスを作成できます。セグメントは、特定のデータベース・デバイスまたは複数のデータベース・デバイスの集合を指します。セグメント内にインデックスを作成する前に、使用できるセグメントのリストをシステム管理者またはデータベース所有者から入手してください。パフォーマンス上の理由またはその他の考慮事項によって、ある特定のセグメントが既に特定のテーブルまたはインデックスに割り付けられている可能性があるためです。

複数のカラムへのインデックス付け：複合インデックス

指定したすべてのカラム内の結合した値に対して複合インデックスを作成するには、複数のカラム名を指定します。

複合インデックスは、2 つ以上のカラムを 1 つの単位として検索するのが最良であるときに使用します。たとえば、authors テーブルで作家の名前と姓を検索する場合などです。複合インデックスに指定するカラムを、テーブル名の後にカッコで囲んでソートの優先順位に従ってリストします。

```
create index auth_name_ind
on authors(au_fname, au_lname)
```

複合インデックスのカラムは、create table 文のカラムと同じ順序である必要はありません。たとえば、au_lname と au_fname の順序を反対にしても構いません。

1 つの複合インデックスには、最大 31 カラムを指定できます。しかし、複合インデックスに指定するすべてのカラムは、同じテーブルのものでなければなりません。『パフォーマンス&チューニング・シリーズ：物理データベースのチューニング』の「第 4 章 テーブルとインデックス・サイズ」を参照してください。

関数ベース・インデックスを使用したインデックス付け

関数ベース・インデックスは、インデックス・キーとして 1 つ以上の式を含みます。インデックスは関数や式で直接作成できます。

計算カラムと同様、関数ベース・インデックスは、集約的なデータ操作を頻繁に必要とするユーザ定義の順序付けおよび意思決定支援システム (DSS: Decision Support System) アプリケーションで役立ちます。関数ベース・インデックスを使用すると、このようなアプリケーションでのタスクが簡略化され、パフォーマンスが向上します。

計算カラムの詳細については、「計算カラム」(319 ページ)を参照してください。

関数ベース・インデックスと計算カラムは、どちらも式でインデックスを作成できるという点では似ています。

ただし、以下の点が大きく異なります。

- 関数ベース・インデックスでは、式を直接インデックス化できる。最初にカラムを作成しない。
- 関数ベース・インデックスは deterministic であることが必要で、計算カラムとは異なり、グローバル変数を参照できない。
- 計算カラムのクラスタード・インデックスは作成できるが、関数ベースのクラスタード・インデックスは作成できない。

create index を実行する前に、データベース・オプション select into を有効にする必要があります。

```
sp_dboption <dbname>, 'select into', true
```

『リファレンス・マニュアル：コマンド』および『リファレンス・マニュアル：プロシージャ』を参照してください。

unique オプションの使用

ユニーク・インデックスでは、null 値も含めて同じインデックス値を持つ 2 つのローは許可されません。既にデータが存在する場合は、インデックスの作成時に、または insert コマンドや update コマンドによってデータを追加または修正するたびに、システムは重複値がないかどうかをチェックします。

データ自体がユニークな特性を持つ場合に限って、ユニーク・インデックスを指定する意味があります。たとえば、last_name カラムに対してユニーク・インデックスを作成しても意味がありません。これは、ローの数が数百程度のテーブルであっても “Smith” や “Wong” という名前が複数存在する可能性があるからです。

しかし、社会保険番号を保管したカラムにユニーク・インデックスを作成することは効果的です。これは、データの特性がユニークで、各人がそれぞれ異なる社会保険番号を持っているからです。さらに、ユニーク・インデックスは整合性をチェックするためにも役立ちます。たとえば、同じ社会保険番号が存在する場合、データ入力時のエラーか、または政府側の処理の誤りであることがわかります。

同じ値が重複して存在するデータに対してユニーク・インデックスを作成しようとすると、コマンドがアボートされ、Adaptive Server は重複する最初の値を示すエラー・メッセージを表示します。null 値を含むローが複数存在するカラムに対しても、ユニーク・インデックスは作成できません。この場合も、値が重複したときと同じ処理が行われます。

ユニーク・インデックスを持つデータを変更する場合には、コマンドを実行したときに ignore_dup_key オプションを使用したかどうかによって結果が異なります。「ignore_dup_key オプションの使用」(423 ページ) を参照してください。

複合インデックスでは、unique キーワードを使用できます。

ユニークでないインデックスにおける IDENTITY カラムの指定

identity in nonunique index データベース・オプションは、テーブルのインデックス・キーに対し、自動的に IDENTITY カラムを含めます。これによって、テーブルに作成したインデックスはすべてユニークになります。このオプションは、論理的にはユニークでないインデックスを内部的にユニークにし、それらのインデックスを使用して更新可能なカーソルと独立性レベル 0 の読み込みを処理できるようにします。

identity in nonunique indexes データベース・オプションを有効にするには、次のように入力します。

```
sp_dboption pubs2, "identity in nonunique index", true
```

テーブルに IDENTITY カラムが事前に存在している必要があります。それには、create table 文で指定するか、テーブルを作成する前に auto identity データベース・オプションを true に設定します。

ユニークでないインデックスを持つテーブルでカーソルと独立性レベル 0 の読み込みを使用するには、identity in nonunique index を使用します。ユニークなインデックスが存在していれば、カーソルに対して次回 fetch を実行したときにカーソルが確実に正しいローに位置付けられます。

たとえば、identity in nonunique index と auto identity データベース・オプションをいずれも true に設定してから、インデックスを持たないテーブルを次のようにして作成したとします。

```
create table title_prices
(title varchar(80)  not null,
price  money        null)
```

sp_help を使用すると、このテーブル内に SYB_IDENTITY_COL という IDENTITY カラムがあることが表示されます。このカラムは、auto identity データベース・オプションの設定によって自動的に作成されたものです。title カラムにインデックスを作成する場合、そのインデックスに IDENTITY カラムが自動的に組み込まれていることを確認するには、sp_helpindex を使用します。

インデックス付きカラム値の昇順と降順

asc (昇順) と desc (降順) キーワードを使用して、インデックスの各カラムにソート順を割り当てることができます。デフォルトのソート順は、昇順です。

カラムにインデックスを作成すると、カラムがクエリの order by 句で指定された順序と同じになるため、クエリの処理中にカラムのソート処理が省略されます。次に、Orders テーブルにインデックスを作成する例を示します。このインデックスには 2 つのカラムがあり、最初の customer_ID は昇順、次の date は降順でソートされます。これは、最も新しい注文を先にリストします。

```
create index nonclustered cust_order_date
    on Orders
    (customer_ID asc,
    date desc)
```

fillfactor、*max_rows_per_page*、*reservepagegap* の使用

fillfactor、max_rows_per_page、reservepagegap は領域を管理するプロパティで、テーブルとインデックスに適用され、物理ページを埋める方法を決定します。『リファレンス・マニュアル：コマンド』を参照してください。表 13-1 に、インデックスの領域を管理するプロパティについての情報を示します。

表 13-1: インデックスの領域を管理するプロパティ

プロパティ	説明	使用	コメント
fillfactor	インデックスの作成時に充填されるページ上の領域の割合を指定する。fillfactor が 100% より小さいと、ただちにページ分割を発生せずに、ページの挿入時に領域を残す。 利点： • 初めはページ分割の発生が少ない。 • より多くのページに、より少ないローしかないため、ページの競合が減る。	データオンリー・ロック・テーブルのクラスタード・インデックスのみに適用。	fillfactor パーセンテージは、インデックスがすでにデータのあるテーブルに作成されたときだけ使用される。テーブルが作成された後のページや挿入には適用されない。 fillfactor の指定がない場合、システムワイドのデフォルト fillfactor が使用される。最初に 100% と設定しても、sp_configure を使用して変更できる。
max_rows_per_page	ページごとに許可されるローの最大数を指定する。 利点： • ページ当たりのローの数を制限し、ページ数を増やすことによって、ページの競合を減らすことができる。	全ページロック・テーブルにのみ適用される。 このプロパティに設定できる最大値は 256。	max_rows_per_page は、インデックスの作成後、常に適用される。指定がない場合、デフォルトではページにできるだけの多くのローが充填される。
reservepagegap	エクステント割り付け時に空にしておくページの数を決定する。たとえば、16 の reservepagegap は、エクステント割り付け時に 2 エクステントの 16 ページのうち、1 ページを空にすることを意味する。 利点： • ローの転送を減らし、reorg rebuild の実行やインデックスの再作成などの管理作業の頻度を減らすことができる。	すべてのロック・スキームのページに適用される。	reservepagegap の指定がない場合、エクステント割り付け時に空になるページはない。

次の文はインデックスに 65% の fillfactor を設定し、各エクステント割り付け時に 1 つの空ページを作成する reservepagegap を設定します。

```
create index postalcode_ind2
    on authors (postalcode)
    with fillfactor = 10, reservepagegap = 8
```

計算カラムのインデックス

結果のデータ型にインデックスを作成できる場合は、通常カラムと同じように計算カラムのインデックスを作成できます。計算カラム・インデックスを使用すると、XML、text、image、Java クラスのような複雑なデータ型に対してインデックスを作成できます。

たとえば、以下のコード例は、通常カラムと同じように、計算カラムのクラスタード・インデックスを作成します。

```
CREATE CLUSTERED INDEX name_index on parts_table(name_order)
CREATE INDEX adt_index on parts_table(version_order)
CREATE INDEX xml_index on parts_table(spec_index)
CREATE INDEX text_index on parts_table(descr_index)
```

インデックスを作成または更新する場合、Adaptive Server は計算カラムを評価し、その結果を使用してインデックスを構築または更新します。

関数ベースのインデックス

関数ベースのインデックス機能を使用すると、関数と式でインデックスを直接作成できます。計算カラム・インデックスと同様、この機能は、ユーザ定義の順序付けと DDS アプリケーションに役立ちます。

次の例は、テーブルの 3 つのカラムを使用して、汎用インデックス・キーでインデックスを作成します。

```
CREATE INDEX generalized_index on parts_table
     (general_key(part_no,listPrice, part_no>>version)
```

状況によっては、個々のカラムのインデックスを作成できない場合に、複数カラムの複合値を返すユーザ定義関数を呼び出して「汎用インデックス・キー」を作成できます。

クラスタード・インデックスとノンクラスタード・インデックスの使用

クラスタード・インデックスでは、Adaptive Server は最新のデータを基準として、ローの物理的な順序と論理的な順序 (インデックスの順序) が同じになるようにローをソートします。クラスタード・インデックスの下位レベルまたは「リーフ・レベル」には、テーブルの実際のデータ・ページが含まれています。ノンクラスタード・インデックスはクラスタード・インデックスが作成されると自動的に再構築されるため、クラスタード・インデックスを作成してから、ノンクラスタード・インデックスを作成します。

クラスタード・インデックスは、テーブルごとに 1 つだけ持つことができます。これは通常、「プライマリ・キー」に対して作成します。プライマリ・キーは、ローをユニークに識別するためのカラムです。

論理的には、プライマリ・キーはデータベースの設計によって決定されます。create table 文または alter table 文で primary key 制約を指定すると、インデックスを作成してテーブルのカラムにプライマリ・キーの属性を適用できます。制約についての情報を表示するには、sp_helpconstraint を使用します。

また、sp_primarykey、sp_foreignkey、sp_commonkey を使用して、プライマリ・キー、外部キー、共通キー (頻繁にジョインされるキーのペア) を明示的に定義することもできます。ただし、これらのプロシージャは、キーの関係を強制しません。

sp_helpkey を使用して定義されたキーの情報を表示したり、sp_helpjoins を使用してジョインの候補になるカラムの情報を表示できます。『リファレンス・マニュアル：プロシージャ』を参照してください。プライマリ・キーと外部キーの定義については、「第 20 章 トリガ：参照整合性」を参照してください。

ノンクラスタード・インデックスを使用すると、ローの物理的な順序はインデックスの順序と一致しません。ノンクラスタード・インデックスのリーフ・レベルには、データ・ページのローを示すポインタが含まれています。厳密には、各リーフ・ページにはインデックス値とその値を持つローに対するポインタが含まれています。つまり、ノンクラスタード・インデックスのインデックス構造とデータ自体との間に、もう 1 つのレベルが存在します。

ノンクラスタード・インデックスは 1 つのテーブルにつき最大 249 個まで作成でき、異なるソート順でデータにアクセスできます。

クラスタード・インデックスを使用したデータ検索は、通常ノンクラスタード・インデックスより高速です。さらに、連続したキー値を持つローを多数検索する場合、つまり、値の範囲を検索するカラムのローを頻繁に検索する場合は、クラスタード・インデックスが効果的です。範囲に当てはまる最初の「キー値」を持つローが検索されると、続くインデックス値を持つローは物理的に隣接して存在するため、それ以降の検索は必要なくなります。

clustered と nonclustered キーワードのいずれも使用しない場合、Adaptive Server はノンクラスタード・インデックスを作成します。

次に、titles テーブルの title_id カラムに titleidind インデックスを作成するコマンドを示します。このコマンドを実行する前に、次のようにインデックスを削除しておきます。

```
drop index titles.titleidind
```

次に、クラスタード・インデックスを作成します。

```
create clustered index titleidind
on titles(title_id)
```

「第 8 章 データベースおよびテーブルの作成」で作成した friends_etc テーブルに保管されている人のデータを、郵便番号で頻繁にソートすることが予測される場合は、次のようにして postalcode カラムにノンクラスタード・インデックスを作成します。

```
create nonclustered index postalcodeind
on friends_etc(postalcode)
```

この場合、同じ郵便番号を持っている人が複数存在する可能性があるため、ユニーク・インデックスを作成しても意味がありません。郵便番号がプライマリ・キーではないので、クラスタード・インデックスの作成も適切ではありません。

friends_etc にあるクラスタード・インデックスは、次のように個人の姓と名前のカラムによる複合インデックスになります。

```
create clustered index nmind
on friends_etc(pname, sname)
```

セグメント上のクラスタード・インデックスの作成

create index コマンドを使用すると、指定したセグメントにインデックスを作成できます。クラスタード・インデックスとそのデータ・ページは定義によって同じリーフ・レベルに設定されているので、clustered インデックスを作成して on segment_name 拡張構文を使用すると、テーブルは作成されたデバイスから指定したセグメントへ移動します。

パフォーマンス上の理由で特定のセグメントが予約されている場合があるので、セグメントにテーブルまたはインデックスを作成する前に、システム管理者またはデータベース所有者に確認してください。

インデックス・オプションの指定

インデックス・オプションには、ignore_dup_key、ignore_dup_row、allow_dup_row があります。これらは、insert コマンドまたは update コマンドで、重複キーや重複ローが生成された場合の動作を制御します。表 13-2 に、使用するオプションをインデックスのタイプごとに示します。

表 13-2: インデックス・オプション

インデックス・タイプ	オプション
クラスタード	ignore_dup_row | allow_dup_row
ユニーク・クラスタード	ignore_dup_key
ノンクラスタード	なし
ユニーク・ノンクラスタード	なし

ignore_dup_key オプションの使用

ユニーク・インデックスがあるカラムに、重複する値を挿入しようとすると、コマンドはキャンセルされます。これは、ユニーク・インデックスを作成するコマンドに ignore_dup_key オプションを指定すると、避けることができます。

ユニーク・インデックスは、クラスタード・インデックスまたはノンクラスタード・インデックスとして作成できます。データを入力するとき、重複キーを挿入しようとするたびにエラー・メッセージが表示され、入力がキャンセルされます。この挿入がキャンセルされると、その時点でアクティブだったトランザクションは、update や insert コマンドの実行がなかったかのように継続できます。重複しない値の場合は、正常に挿入されます。

ignore_dup_key の設定にかかわらず、既に重複する値が含まれているカラムにはユニーク・インデックスを作成できません。作成しようとすると、Adaptive Server はエラー・メッセージと重複する値のリストを表示します。重複する値を削除してから、カラムにユニーク・インデックスを作成する必要があります。

次に、ignore_dup_key オプションの使用例を示します。

```
create unique clustered index phone_ind
on friends_etc(phone)
with ignore_dup_key
```

ignore_dup_row オプションと *allow_dup_row* オプションの使用

ignore_dup_row オプションと allow_dup_row オプションは、ユニークでないクラスタード・インデックスを作成するのに指定します。これらのオプションは、ユニークでないノンクラスタード・インデックスを作成する場合には適切ではありません。Adaptive Server ノンクラスタード・インデックスは、ユニークなロー識別番号を内部的に各ローに付加するため、同じデータ値を持つローでも重複ローは問題にはなりません。

ignore_dup_row と allow_dup_row は同時には使用できません。

ユニークでないクラスタード・インデックスを作成した場合、重複キーは作成できますが、重複ローは allow_dup_row オプションを指定しないかぎり作成されません。

allow_dup_row を設定すると、重複ローを含むテーブルに対してユニークでないクラスタード・インデックスを新しく作成できます。この後、重複ローを insert または update できます。

あるテーブルの任意のインデックスがユニークである場合、ユニーク要件(最も厳しい要件) は allow_dup_row オプションよりも優先します。このように、allow_dup_row は、ユニークでないインデックスを持つテーブルに対してだけ適用されます。テーブルのいずれかのカラムに対してユニーク・クラスタード・インデックスが作成されている場合には、このオプションを使用できません。

ignore_dup_row オプションを使用すると、データのバッチから重複が取り除かれます。重複ローとなる値が入力されると、Adaptive Server はそのローを無視し、このとき発行された insert コマンドまたは update コマンドをキャンセルし、情報エラー・メッセージを表示します。この挿入がキャンセルされた後、その時点でアクティブだったトランザクションは、コマンドの実行がなかったかのように継続することができます。重複しないローは、正常に挿入されます。

ignore_dup_row は、ユニークでないインデックスを持つテーブルに対してだけ適用されます。テーブルのいずれかのカラムに対してユニーク・インデックスが作成されている場合には、このキーワードを使用できません。

表 13-3 に、重複ローを含むテーブルにユニークでないクラスタード・インデックスを作成しようとした場合と、重複ローをテーブルに挿入しようとした場合の allow_dup_row オプションと ignore_dup_row オプションの動作を示します。

表 13-3: インデックスの重複ロー・オプション

オプション	重複ローが存在する場合	重複ローを入力した場合
どのオプションも指定しない場合	create index コマンドは失敗	コマンドは失敗
allow_dup_row を設定	コマンドは正常に完了	コマンドは正常に完了
ignore_dup_row を設定	インデックスは作成されるが、重複ローは削除される。エラー・メッセージが表示される。	重複ローの挿入または更新はされない。エラー・メッセージが表示される。トランザクションは正常に完了する。

sorted_data オプションの使用

テーブル内のデータの順序がソート済みの場合、create index コマンドの sorted_data オプションを指定すると、インデックスを高速に作成できます。たとえば、ソート済みのデータを bcp を使って空のテーブルにコピーする場合に便利です。インデックスの作成速度は、サイズの大きいテーブルを対象とするときに大幅に向上し、1 GB を超えるテーブルでは作成速度は数倍速くなります。

sorted_data をデータがソートされていないテーブルに指定すると、エラー・メッセージが表示され、コマンドはアボートされます。

sorted_data によって高速化するのは、クラスタード・インデックスかユニーク・ノンクラスタード・インデックスを作成する場合だけです。ただし、重複キーが存在しない場合にだけ、ユニークでないノンクラスタード・インデックスを作成するときにも有効です。重複キーを持つローが存在する場合は、エラー・メッセージが表示され、コマンドはアボートされます。

他の create index オプションには、sorted_data を指定した場合でもデータのソートが必要なものがあります。『リファレンス・マニュアル：コマンド』を参照してください。

on segment_name オプションの使用

on segment_name 句は、インデックスを作成するデータベース・セグメント名を指定します。ノンクラスタード・インデックスは、データ・ページとは異なるセグメントに作成できます。次に例を示します。

```
create index titleind
on titles(title)
on seg1
```

クラスタード・インデックスを作成するときに segment_name を指定すると、作成したインデックスを持つテーブルは、指定したセグメントまで移動します。パフォーマンス上の理由で特定のセグメントが予約されている場合があるので、セグメントにテーブルまたはインデックスを作成する前に、システム管理者またはデータベース所有者に確認してください。

インデックスの削除

drop index コマンドは、データベースからインデックスを削除します。

このコマンドを使用すると、Adaptive Server は指定されたインデックスをデータベースから削除し、インデックスが使用していた記憶領域を再利用できるようにします。

インデックスを削除できるのはその所有者だけで、drop index パーミッションは他のユーザには譲渡できません。drop index コマンドは、master データベースまたはユーザ・データベースのシステム・テーブルに対しては使用できません。

クエリのときにほとんど、またはまったく使用されないインデックスは、削除したい場合があります。

たとえば、friends_etc テーブルの phone_ind インデックスを削除するには、次のコマンドを実行します。

```
drop index friends_etc.phone_ind
```

エンディアン・タイプの異なるプラットフォーム間で load database を実行した後には、sp_post_xpload を使用してインデックスをチェックして再構築します。

テーブルに存在するインデックスの確認

テーブルに存在するインデックスを確認するには、sp_helpindex を使用します。次に、friends_etc テーブルのヘルプ・レポートを示します。

```
sp_helpindex friends_etc

index_name      index_keys    index_description    index_max_rows_per_page
-------------   ----------   ----------------- --------------------------
nmind           pname,sname   clustered                                 0
postalcodeind   postalcode    nonclustered                              0

index_fillfactor  index_reservepagegap  index_created        index_local
----------------- --------------------  ---------------      ------------
               0                     0  May 24 2005 1:49PM   Global Index
               0                     0  May 24 2005 1:49PM   Global Index

(2 rows affected)

index_ptn_name            index_ptn_seg
------------------        ---------------
nmind_1152004104          default
postalcodeind_1152004104  default
(2 rows affected)
```

sp_help は、レポートの終わりに sp_helpindex を実行します。

sp_statistics は、テーブル上のインデックスのリストを返します。次に例を示します。

```
sp_statistics friends_etc

table_qualifier               table_owner
```

```
         table_name                      non_unique
         index_qualifier                 index_name
         type   seq_in_index column_name                      collation
         cardinality pages
-------------------------------- --------------------------------
        -------------------------------- ----------
        -------------------------------- --------------------------------
        ------ ------------ -------------------------------- ---------
        ----------- -----------
pubs2                           dbo
        friends_etc                           NULL
        NULL                            NULL
             0          NULL NULL                              NULL
                   0           1
pubs2                             dbo
         friends_etc                          1
         friends_etc                    nmind
            1           1 pname                        A
                   0           1
pubs2                             dbo
         friends_etc                           1
         friends_etc                     nmind
               1            2 sname                         A
                   0            1
pubs2                             dbo
         friends_etc                          1
         friends_etc                    postalcodeind
             3           1  postalcode                      A
                NULL       NULL

(4 rows affected)

table_qualifier table_owner table_name index_qualifier index_name
       non_unique_type seq_in_index column_name collation index_id
       cardinality pages status status2
---------------- ---------  ---------- --------------- -----------
       ---------------- ------------ ----------- --------- ---------
       ----------- ----- ------ --------
pubs2            dbo         friends_etc friends_etc     nmind
                   1     1              1 pname      A
                     0          1        16       0
pubs2            dbo         friends_etc friends_etc     nmind
                   1     1              2 sname      A
                   0          1        16       0
pubs2            dbo         friends_etc friends_etc     postalcodeind


                   1     3              1 postalcode A
                  NULL          NULL       0       0
pubs2             dbo         friends_etc NULL            NULL
                  NULL     0      NULL NULL    NULL    0
```

```
                    0           1           0       0
(4 rows affected)
(return status = 0)
```

さらに、テーブル名の後に “1” を指定して sp_spaceused を使用すると、テーブルとそのインデックスが使用している記憶領域の容量がレポートされます。次に例を示します。

```
sp_spaceused friends_etc, 1

index_name           size       reserved   unused
-------------------- ---------- ---------- ----------
nmind                2 KB       32 KB      28 KB
postalcodeind        2 KB       16 KB      14 KB

name         rowtotal   reserved     data    index_size      unused
------------ ---------- ------------ ------- --------------- --------
friends_etc  1          48 KB        2 KB    4 KB            42 KB

(return status = 0)
```

インデックスに関する統計値の更新

update statistics コマンドを使用すると、Adaptive Server がクエリの処理に使用する最も適切なインデックスを決定するときに役立ちます。インデックス内のキー値の分布に関する最新のデータを保持します。update statistics コマンドは、インデックス付きのカラムで大量のデータが追加、変更、削除された場合に使用します。

コンポーネント統合サービスを有効にすると、update statistics はリモート・テーブルの正確な分布統計値を生成できます。『コンポーネント統合サービス・ユーザーズ・ガイド』を参照してください。

update statistics コマンドの実行パーミッションは、デフォルトではテーブル所有者に与えられ、他のユーザには譲渡できません。構文は次のとおりです。

update statistics *table_name* [*index_name*]

インデックス名を指定しない場合は、指定したテーブルにあるすべてのインデックスの分布統計値が更新されます。インデックス名を指定した場合は、該当するインデックスの統計値だけが更新されます。

インデックス名を検索するには、sp_helpindex を使用します。『リファレンス・マニュアル：プロシージャ』を参照してください。

テーブル内のすべてのインデックスの統計を更新するには、テーブル名を入力します。

```
update statistics authors
```

au_id カラムに対するインデックスの統計値だけを更新するには、次のように入力します。

```
update statistics authors auidind
```

Transact-SQL ではインデックス名がデータベース内でユニークでなくてもよいため、インデックスが関連付けられているテーブルの名前を指定する必要があります。既存のデータにインデックスを作成すると、Adaptive Server が自動的に update statistics を実行します。

コマンドの実行によってシステムに支障をきたさないように、サイトで都合のよい時間帯に update statistics が自動実行されるように設定できます。

第 14 章

データのデフォルトとルールの定義

「デフォルト」とは、ユーザがカラムの値を明示的に入力しない場合に、Adaptive Server がそのカラムに挿入する値のことです。データベース管理においては、「ルール」を使用すると、特定のカラムまたは指定されたユーザ定義データ型を持つカラムに入力できる値と入力できない値が指定できます。デフォルトとルールによって、データベース全体に渡ったデータの整合性を維持することができます。

デフォルトとルールの機能

テーブル・カラムまたはユーザ定義データ型の値を指定すると、ユーザが値を明示的に入力しない場合に、自動的にこの値が挿入されます。たとえば、“???” や “fill in later” のような値を持つデフォルトを作成できます。また、テーブル・カラムまたはユーザ定義データ型にルールを定義して、ユーザが入力できる値の種類を限定することもできます。

リレーショナル・データベース管理システムでは、すべてのデータ要素に値 (または null 値) が必要です。「第 8 章 データベースおよびテーブルの作成」に説明されているように、null 値を受け入れないカラムもあります。そうしたカラムには、null 以外の値で、ユーザが明示的に指定する値か、Adaptive Server のデフォルト値のいずれかを入力する必要があります。

ルールは、カラムのデータ型ではカバーできないデータの整合性を強化します。特定のカラム、複数の特定のカラム、または特定のユーザ定義データ型に対して、ルールを割り当てることができます。

ユーザが値を入力するたびに、Adaptive Server は特定のカラムにバインドされた最新のルールとその値をチェックします。ルールの作成前およびバインド前に入力されたデータはチェックされません。

デフォルトの句の共有可能なインライン・デフォルト・オブジェクトを作成して、同じデフォルト・オブジェクトを複数のテーブルおよびカラムに自動的に使用できます。「インライン・デフォルトの共有」(443 ページ) を参照してください。

デフォルトとルールの代わりに、default 句と create table 文の check 整合性制約を使用して、同じようなタスクのいくつかを実行できます。しかし、このような項目は各テーブルに固有のもので、別のテーブルのカラムやユーザ定義データ型にはバインドはできません。整合性制約の詳細については、「第 8 章 データベースおよびテーブルの作成」を参照してください。

デフォルトの作成

デフォルトは、テーブルにデータを入力する前と後のどちらでも、作成または削除できます。デフォルトを作成する一般的な手順は、次のとおりです。

1 create default を使用してデフォルトを定義します。

2 sp_bindefault を使用して、該当するテーブル・カラムまたはユーザ定義データ型にデフォルトをバインドします。

3 データを挿入して、バインドしたデフォルトをテストします。

デフォルトは drop default を使用して削除でき、sp_unbinddefault を使って関連付けを削除できます。

デフォルトの作成とバインドを行う場合、次のことに注意します。

- カラムの大きさがデフォルトに適しているかどうかを確認すること。たとえば、char(2) カラムには、“nobody knows yet” のような 17 バイトの文字列は挿入できません。
- ユーザ定義データ型にデフォルトを挿入し、そのタイプの個々のカラムに別のデフォルトを挿入する場合は注意すること。最初にデータ型のデフォルトをバインドし、次にカラムのデフォルトをバインドすると、指定のカラムに対してだけはユーザ定義データ型ではなく、カラムのデフォルトが使用されます。ユーザ定義データ型のデフォルトは、そのデータ型を持つ他のすべてのカラムにバインドされます。

  ただし、タイプに従ってデフォルトを持つカラムに他のデフォルトをバインドすると、そのカラムはデータ型にバインドされたデフォルトの影響を受けません。この問題の詳細については、「デフォルトのバインド」(434 ページ) を参照してください。
- デフォルトとルールとの矛盾に注意すること。デフォルト値がルールで許可されているかどうかを確認します。許可されていない場合、ルールによってデフォルトが削除されることがあります。

たとえば、ルールが 1 ～ 100 のエントリを許可していて、デフォルトが 0 に設定されている場合は、ルールによってデフォルト・エントリが拒否されます。デフォルトかルールのいずれかを変更してください。

create default 構文

create default の構文は、次のとおりです。

```
create default [owner.]default_name
    as constant_expression
```

デフォルト名は、識別子の規則に適合していなければなりません。現在のデータベース内でのみ、デフォルトを作成できます。

データベース内では、デフォルト名はユーザごとにユニークにする必要があります。たとえば、phonedflt というデフォルトを 2 つ作成することはできません。ただし、既に dbo.phonedflt が存在する場合でも、所有者の名前によって区別できるため、“guest” として phonedflt を作成できます。

別の例：friends_etc の city カラムと、できれば他のカラムまたはユーザ・データ型とともに使用できる “Oakland” のデフォルト値を作成するとします。デフォルトを作成するには、次のように入力します。

```
create default citydflt
as "Oakland"
```

この例に従うと、ユーザ個人のテーブルに入力する予定の人に対して、あらゆる都市名を使用できるようになります。

文字やデータ定数は引用符で囲みます。通貨、整数、浮動小数点の定数は、引用符で囲む必要はありません。バイナリ・データの前には “0x”、通貨データの前にはドル記号 ($) など対象地域の論理的なデフォルト通貨を表す通貨記号が必要です。デフォルト値は、カラムのデータ型と互換性があることが必要です。たとえば、数値カラムのデフォルトとして “none” は使用できませんが、0 は適しています。

通常、テーブルを作成したらデフォルト値を入力します。しかし、1 つまたは複数のカラムに同じ値を持つローを多く入力するセッションでは、開始前にセッションに合ったデフォルトを作成することもできます。

注意　create table をデフォルト宣言付きで発行した後で、その同じバッチまたはプロシージャ内でテーブルにデータを挿入することはできません。create 文と insert 文を 2 つの異なるバッチまたはプロシージャに分けるか、execute を使用してアクションを別々に実行してください。

デフォルトのバインド

デフォルトを作成した後は、sp_bindefault を使用してカラムまたはユーザ定義データ型にデフォルトをバインドします。たとえば、次のデフォルトを作成するとします。

```
create default advancedflt as "UNKNOWN"
```

デフォルトを該当するカラムまたはユーザ定義データ型にバインドします。

```
sp_bindefault advancedflt, "titles.advance"
```

titles テーブルの advance カラムにエントリを作成しない場合だけ、デフォルトが有効になります。エントリを入力しないことは、null 値を入力することとは異なります。デフォルトは、特定カラム、複数のカラム、またはデータベース内の指定されたユーザ定義データ型を持つすべてのカラムに関連付けることができます。

注意 デフォルトを取得するには、デフォルトを持つカラムの含まれないカラム・リストを使用して、insert コマンドまたは update コマンドを発行する必要があります。

適用される制限は、次のとおりです。

- デフォルトは新しいローだけに適用されます。既存のローにさかのぼって変更することはできません。デフォルトは、ユーザが何も入力しなかった場合にのみ使用されます。ユーザがカラムの値 (null も含む) を入力した場合は、デフォルトは適用されません。
- デフォルトをシステム・データ型にバインドすることはできません。
- timestamp カラムには、デフォルトをバインドできません。これは、Adaptive Server が timestamp カラムの値を自動的に生成するからです。
- システム・テーブルには、デフォルトをバインドできません。
- IDENTITY カラムまたは IDENTITY プロパティを持つユーザ定義データ型にはデフォルトをバインドできますが、Adaptive Server はそのデフォルトを無視します。カラムに値を指定しないでテーブルにローを挿入すると、Adaptive Server は最後に割り当てた値より 1 大きい値を割り当てます。
- カラムにデフォルトがすでに存在する場合、そのデフォルトを削除してから、新しいデフォルトをバインドします。sp_bindefault で作成したデフォルトを削除するには、sp_unbindefault を使用します。create table で作成したデフォルトを削除するには、alter table を使用します。

friends_etc の city カラムに citydflt をバインドするには、次のように入力します。

```
sp_bindefault citydflt, "friends_etc.city"
```

埋め込み句読点であるピリオドがあるので、テーブルおよびカラム名は引用符で囲む必要があります。

データベース内の各テーブルの city カラムすべてに特殊なデータ型を作成し、さらに citydflt をそのデータ型にバインドすると、市名が該当する場合、“Oakland” が表示されます。たとえば、ユーザ・データ型が citytype の場合に、citydflt をバインドする方法は次のようになります。

```
sp_bindefault citydflt, citytype
```

既存のカラムまたは特殊なユーザ定義データ型が、新しいデフォルトを継承しないようにするには、デフォルトをユーザ定義データ型にバインドするときに futureonly パラメータを使用します。しかし、デフォルトをカラムにバインドするときは、futureonly を使用しないでください。次に、新しいデフォルト “Berkeley” を作成し、それを新しいテーブル・カラムが使用するデータ型 citytype にバインドする方法を示します。

```
create default newcitydflt as "Berkeley"
sp_bindefault newcitydflt, citytype, futureonly
```

“Oakland” は、citytype を使用する既存のテーブル・カラムのデフォルトとして表示が継続されます。

テーブルにあるほとんどの人が同じ郵便番号の地域に住んでいる場合、デフォルトを作成してデータ・エントリの時間を短縮できます。次に、オークランド地区に適切なデフォルトと、そのバインドを示します。

```
create default zipdflt as "94609"
sp_bindefault zipdflt, "friends_etc.postalcode"
```

次に、sp_bindefault の完全な構文を示します。

sp_bindefault *defname, objname* [, futureonly]

defname は、create default で作成されたデフォルトの名前で、*objname* はデフォルトをバインドする対象のテーブルやカラムの名前か、またはユーザ定義データ型の名前です。このパラメータが *table.column* の形式になっていない場合は、ユーザ定義データ型であると判断されます。

オプションの futureonly パラメータを使用して、指定したユーザ定義データ型の既存のカラムがデフォルトを継承するのを防止しないかぎり、そのユーザ定義型のカラムはすべて、指定したデフォルトに関連付けられます。

注意 デフォルトはカラムにバインドできず、同じバッチの実行中に使用することもできません。sp_bindefault は、デフォルトを起動する insert 文と同じバッチ内で実行できません。

デフォルトを作成した後、そのデフォルトを説明する「ソース・テキスト」が syscomments システム・テーブルの text カラムに保管されます。この情報は削除しないでください。削除すると、将来 Adaptive Server をアップグレードするときに問題が発生する場合があります。代わりに、sp_hidetext を使用して、syscomments 内でテキストを暗号化します。『リファレンス・マニュアル：プロシージャ』および「コンパイル済みオブジェクト」(3 ページ) を参照してください。

デフォルトのバインド解除

デフォルトのバインド解除とは、特定のカラムやユーザ定義データ型からそのデフォルトとの関連付けを削除することを意味します。バインド解除されたデフォルトは、そのままデータベースに保持され、今後も使用できます。デフォルトとカラムまたはデータ型とのバインドを解除するには、sp_unbindefault を使用します。

次に、friends_etc テーブルの city カラムから現在のデフォルトをバインド解除する例を示します。

```
execute sp_unbindefault "friends_etc.city"
```

ユーザ定義データ型 citytype からデフォルトのバインドを解除するには、次のように入力します。

```
sp_unbindefault citytype
```

次に、sp_unbindefault の完全な構文を示します。

sp_unbindefault *objname* [, futureonly]

指定した *objname* パラメータが *table.column* の形式になっていない場合、Adaptive Server はユーザ定義データ型であると判断します。ユーザ定義データ型からデフォルトのバインドを解除すると、デフォルトはそのタイプのすべてのカラムからバインド解除されます。ただし、オプションの futureonly パラメータを使用する場合は、そのデータ型の既存のカラムへのデフォルトのバインドは解除されません。

デフォルトの null 値への影響

カラムを作成するときにそのカラムにデフォルトを関連付けずに not null を指定した場合、ローを挿入するときにそのカラムに値を入力しないと、エラー・メッセージが表示されます。

null カラムのデフォルトを削除すると、そのカラムに値を入力しないでローを追加するたびに、Adaptive Server はそのカラムに null を挿入します。not null カラムのデフォルトを削除すると、ローを追加してそのカラムの値を入力しない場合は、エラー・メッセージが表示されます。

表 14-1 は、デフォルトの有無と null カラムまたは not null カラムの定義との関係を示します。

表 14-1: カラム定義と null デフォルト

カラム定義	ユーザ入力	結果
null とデフォルトが定義されている	値なし	デフォルトを使用
	null 値	null を使用
null が定義され、デフォルトは定義されていない	値なし	null を使用
	null 値	null を使用
not null で、デフォルトが定義されている	値なし	デフォルトを使用
	null 値	エラー
not null が定義され、デフォルトが定義されていない	値なし	エラー
	null 値	エラー

デフォルトの削除

デフォルトをデータベースから削除するには、drop default コマンドを使用します。すべてのカラムとユーザ・データ型からデフォルトのバインドを解除してから、削除を行ってください(「デフォルトのバインド解除」(436 ページ)を参照)。まだバインドされているデフォルトを削除しようとすると、Adaptive Server はエラー・メッセージを表示し、drop default コマンドが失敗します。

次に、citydflt を削除する方法を示します。最初に、バインドを解除します。

```
sp_unbindefault citydft
```

次に、citydft を削除します。

```
drop default citydflt
```

drop default の完全な構文は、次のとおりです。

```
drop default [owner.]default_name
  [, [owner.]default_name] ...
```

デフォルトを削除できるのは、その所有者だけです。『リファレンス・マニュアル：プロシージャ』および『リファレンス・マニュアル：コマンド』を参照してください。

ルールの作成

ルールを使用して、特定のカラムやユーザ定義データ型のカラムに、ユーザが入力できる内容と入力できない内容を指定できます。ルールを作成する一般的な手順は、次のとおりです。

1 create rule を使用してルールを作成します。

2 sp_bindrule を使用してカラムやユーザ定義データ型にルールをバインドします。

3 データを挿入して、バインドしたルールをテストします。insert または update コマンドでテストするだけで、ルールの作成やバインドのエラーを数多く検出できます。

sp_unbindrule を使用するか、またはカラムやデータ型に新しいルールをバインドすることによって、カラムやデータ型からルールをバインド解除できます。

create rule 構文

create rule の構文は、次のとおりです。

```
create rule [owner.]rule_name
    as condition_expression
```

ルール名は、識別子の規則に従います。現在のデータベース内でのみ、ルールを作成できます。

データベース内では、ルール名はユーザごとにユニークにする必要があります。たとえば、socsecrule というルールを 2 つ作成することはできません。しかし、それぞれのルール名は所有者名によって区別されるので、socsecrule という名前のルールを 2 人のユーザが個別に作成することは可能です。

次は、5 つの異なる pub_id 番号とダミー値 (99 のあとに任意の 2 つの整数) を許可するルールの作成例です。

```
create rule pub_idrule
as @pub_id in ("1389", "0736", "0877", "1622", "1756")
or @pub_id like "99[0-9][0-9]"
```

as 句には、最初に “@” が付くルールの引数名とルール自体の定義が含まれます。引数は、update や insert 文の影響を受けるカラム値を参照します。

このルールが pub_id カラムにバインドされるので、引数は @pub_id というわかりやすい名前になっています。どの名前でも引数に使用できますが、最初の文字は “@” にする必要があります。ルールをバインドするカラムまたはデータ型の名前を使用すると、バインドする対象がわかりやすくなります。

ルールの定義には、where 句で有効な式、算術演算子、または like、in、between などの比較演算子を使用できます。しかし、ルールの定義は直接カラムや他のデータベース・オブジェクトを参照できません。データベース・オブジェクトを参照しない組み込み関数は使用できます。

次の例では、入力する値が特定の「状況」を満たさなければならないルールを作成しています。この場合、カラムに入力するそれぞれの値は、数字 “415” で始まり、その後に 7 文字が必要です。

```
create rule phonerule
as @phone like "415_______"
```

入力する友人の年齢が 17 以外の 1 ～ 120 までであるようにするには、次のように入力します。

```
create rule agerule
as @age between 1 and 120 and @age != 17
```

ルールのバインド

ルールを作成した後は、sp_bindrule を使用してカラムまたはユーザ定義データ型にルールをリンクします。

次に、sp_bindrule の完全な構文を示します。

sp_bindrule *rulename*, *objname* [, futureonly]

rulename は、create rule によって作成されたルール名です。*objname* は、ルールがバインドされるテーブルとカラム、またはユーザ定義データ型の名前です。このパラメータが *table.column* の形式になっていない場合は、ユーザ定義データ型であると判断されます。

オプションの futureonly パラメータは、ユーザ定義データ型にルールをバインドするときだけ使用します。指定したユーザ定義データ型のすべてのカラムは、futureonly を指定しないかぎり、指定したルールに関連付けられます。このパラメータは、ユーザ・データ型の既存のカラムにルールが反映されないようにします。指定のユーザ定義データ型と関連するルールを事前に変更していた場合、Adaptive Server はそのユーザ定義データ型の既存のカラム用に変更されたルールを保持します。

ルールを定義したあと、そのルールを説明する「ソース・テキスト」が syscomments システム・テーブルの text カラムに保管されます。この情報は削除しないでください。削除すると、将来 Adaptive Server をアップグレードするときに問題が発生する場合があります。代わりに、sp_hidetext を使用して、syscomments 内でテキストを暗号化します。『リファレンス・マニュアル：プロシージャ』および「コンパイル済みオブジェクト」(3 ページ)を参照してください。

適用される制限は、次のとおりです。

- text、unitext、image、timestamp データ型のカラムにルールをバインドすることはできない。
- システム・テーブルに関するルールは使用できない。

カラムにバインドするルール

ルール名、引用符付きテーブル名、およびカラム名を指定した sp_bindrule を使用して、カラムにルールをバインドします。次のようにして、publishers.pub_id に pub_idrule をバインドします。

```
sp_bindrule pub_idrule, "publishers.pub_id"
```

次の例は、入力するすべての郵便番号の最初の 3 桁を 946 にするためのルールです。

```
create rule postalcoderule946
as @postalcode like "946[0-9][0-9]"
```

次のようにして、friends_etc の postalcode カラムにこのルールをバインドします。

```
sp_bindrule postalcoderule946, "friends_etc.postalcode"
```

同じバッチの実行中にルールを複数のカラムにバインドしたり、使用することはできません。sp_bindrule を、ルールを起動する insert 文と同じバッチ内に指定することはできません。

ユーザ定義データ型にバインドするルール

システム・データ型にルールをバインドすることはできませんが、ユーザ定義データ型にバインドすることはできます。p# というユーザ定義データ型に phonerule をバインドするには、次のように入力します。

```
sp_bindrule phonerule, "p#"
```

ルールの優先度

カラムにバインドされたルールは、ユーザ・データ型にバインドされたルールより優先されます。カラムにルールをバインドすると、そのカラムのユーザ・データ型にバインドされたルールに代わって使用されますが、データ型にルールをバインドしても、そのユーザ・データ型のカラムにバインドされたルールとは置き換わりません。

ユーザ定義データ型にバインドされたルールは、ユーザ定義データ型のデータベース・カラムに値を挿入または更新する場合にだけ有効になります。ルールは変数をテストしないため、同じデータ型のカラムにバインドされたルールによって拒否される値をユーザ定義データ型変数に割り当てないようにしてください。

表 14-2 は、すでにルールが存在するカラムとユーザ・データ型にルールをバインドする場合の優先度を示します。

表 14-2: ルールの優先度

新しいルールのバインド先	古いルールのバインド先	
	ユーザ・データ型	カラム
ユーザ・データ型	古いルールを置き換える	変更なし
カラム	古いルールを置き換える	古いルールを置き換える

一時的に特別な制約が必要なデータを複数のカラムに入力する場合、新しいルールを作成するとデータを簡単にチェックできます。たとえば、friends_etc テーブルの debt カラムにデータを追加するとします。今日記録する負債額は 5 ～ 200 ドルの間の額であることがわかっています。この範囲外の金額を誤って入力しないようにするには、次のようなルールを作成します。

```
create rule debtrule
as @debt = $0.00 or @debt between $5.00 and $200.00
```

@debt ルールの定義では、このカラムに事前に定義されたデフォルトを保持するために 0.00 ドルの入力を許可します。

次のようにして、debtrule を debt カラムにバインドします。

```
sp_bindrule debtrule, "friends_etc.debt"
```

ルールと null 値

カラムが null 値を受け入れるように定義し、その後 null 値を禁止するルールによりこの定義を無効にすることはできません。たとえば、カラム定義で null を指定し、ルールで次のように指定した場合、暗黙的または明示的に null が設定されても違反にはなりません。

```
@val in (1,2,3)
```

ルールで次のように指定されていても、カラム定義によってルールは無効になります。

```
@val is not null
```

ルールのバインド解除

ルールのバインドを解除すると、特定のカラムやユーザ定義データ型からそのルールとの関連付けが削除されます。バインド解除されたルールの定義は、そのままデータベースに保持され、今後も使用できます。

ルールをバインド解除するには、2 つの方法があります。

- ルールとカラムまたはユーザ定義データ型とのバインドを削除するには、sp_unbindefault を使用します。
- sp_bindrule を使用して、新しいルールをカラムかユーザ定義データ型にバインドします。古いルールは自動的にバインド解除されます。

次に、friends_etc.debt から debtrule (または他の現在バインドされているルール) を解除する方法を示します。

```
sp_unbindrule "friends_etc.debt"
```

ルールはデータベースに残ったままですが、friends_etc.debt との関連はありません。

ユーザ定義データ型 p# からルールをバインド解除するには、次のように入力します。

```
sp_unbindrule "p#"
```

次に、sp_unbindrule の完全な構文を示します。

sp_unbindrule *objname* [, futureonly]

使用する *objname* パラメータが "*table.column*" の形式になっていない場合、Adaptive Server はユーザ定義データ型であると判断します。ユーザ定義データ型からルールのバインドを解除すると、ルールはそのタイプのすべてのカラムからバインド解除されます。ただし、次の場合を除きます。

- そのデータ型の既存のカラムからルールがバインド解除されないようにするための、オプションの futureonly パラメータを使用した場合。
- ユーザ定義データ型のカラムのルールが変更されたことによって、その現在の値がバインド解除されるルールと違う場合。

ルールの削除

ルールをデータベースから永久に削除するには、drop rule コマンドを使用します。すべてのカラムとユーザ・データ型からルールのバインドを解除してから、削除を行ってください。まだバインドされているルールを削除しようとすると、Adaptive Server はエラー・メッセージを表示し、drop rule が失敗します。しかし、新しいルールをバインドする場合は、ルールをバインド解除して削除する必要はありません。代わりに、新しいルールをバインドするだけです。

次は、バインド解除した後に phonerule を削除することを示しています。

```
drop rule phonerule
```

次に、drop rule の完全な構文を示します。

drop rule [*owner.*]*rule_name*
[, [*owner.*]*rule_name*] ...

ルールを削除した後、ルールの影響を受けていたカラムに新しく入力されるデータは、これらの制約に関係なく入力されます。既存のデータには影響ありません。

ルールを削除できるのは、その所有者だけです。

デフォルトとルールについての情報の取得

sp_help を、テーブル名を指定して使用すると、カラムにバインドされているルールとデフォルトが表示されます。この例では、ルールとデフォルトを含む pubs2 データベースの authors テーブルに関する情報が表示されます。

```
sp_help authors
```

sp_help は、ユーザ定義データ型にバインドされたルールもレポートします。ユーザ定義データ型 p# にルールがバインドされているかどうかを確認するには、次のように入力します。

```
sp_help "p#"
```

sp_helptext によって、ルールまたはデフォルトの定義 (create 文) が表示されます。

デフォルトまたはルールのソース・テキストを sp_hidetext を使用して暗号化した場合、Adaptive Server はテキストが隠されたことを伝えるメッセージを表示します。『リファレンス・マニュアル：プロシージャ』を参照してください。

システム・セキュリティ担当者が、Adaptive Server を評価済み設定で実行するために sp_configure の allow select on syscomments.text column パラメータをリセットした場合、デフォルトかルールの作成者またはシステム管理者であれば、sp_helptext を使用してデフォルトやルールのテキストを参照することができます。『システム管理ガイド 第 1 巻』の「第 12 章 セキュリティの概要」を参照してください。

インライン・デフォルトの共有

Adaptive Server では、新しいインライン・デフォルトの作成時に、同じユーザに属するデータベース内で同じ値を持つ既存の共有可能なインライン・デフォルトを検索します。検出された場合は、新しいデフォルトを作成する代わりに、このオブジェクトがカラムにバインドされます。しかし、既存の共有可能なインライン・デフォルトが検出されない場合には、新しいデフォルトが作成されます。

Adaptive Server は、同じデータベース内でのみテーブル間でインライン・デフォルトを共有します。

enable functionality group を 0 より大きい数値に設定し、共有インライン・デフォルトを有効にします。『システム管理ガイド 第 1 巻』の「第 5 章 設定パラメータ」を参照してください。

共有可能なインライン・デフォルトの作成

create table、alter table.. . add および alter table. . replace を使用すると、共有可能なインライン・デフォルトを作成できます。

Adaptive Server は、デフォルトの句が create table または alter table コマンドで使用される場合に、共有可能なインライン・デフォルトを自動的に作成して使用します。『リファレンス・マニュアル：コマンド』を参照してください。

たとえば、次のようにこのテーブルを作成するとします。

```
create table my_titles
(title_id char(6),
title    varchar(80),
moddate  datetime default '12/12/2012')
```

次に、同じデフォルトで 2 つ目のテーブルを次のように作成します。

```
create table my_authors2
(auth_id char(6),
title    varchar(80),
moddate  datetime default '12/12/2012')
```

sysobjects では、次の 2 つで共有される単一のデフォルトがレポートされます。

```
select id, name from sysobjects where type = 'D'
 id          name
 ----------- --------------------------
 1791386948  my_titles_moddat_1791386948
```

sp_helpconstraint を使用して、次の共有可能なインライン・デフォルト・オブジェクトの定義を表示します。

```
sp_helpconstraint my_titles
name                              defintion              created
--------------------------------  -------------------    ---------------
my_titles_moddate_1791386948      DEFAULT '12/12/2012'   Dec 6 2010 10:55AM


sp_helpconstraint my_authors2
name                              defintion             created
--------------------------------  -------------------   ----------------
my_titles_moddate_1791386948      DEFAULT '12/12/2012'   Dec 6 2010 10:55AM
```

my_titles および my_authors2 は、同じ内部デフォルト名である my_titles_moddate_1791386948 を表示します。これは、これらのカラムで共有される単一のデフォルト・オブジェクトがあることを示します。また、sp_help は、カラムに関連付けられたデフォルトを示します。

共有インライン・デフォルトのバインド解除

通常のインライン・デフォルトと同様に、共有可能なインライン・デフォルトをカラムから明示的にバインド解除することはできません。Adaptive Server では、drop table または alter table コマンド実行時にそれを自動的にバインド解除またはカラムから削除します。テーブルを削除または変更する場合は、インライン・デフォルトが他のカラムと共有されているかどうか Adaptive Server がチェックします。共有されている場合、共有デフォルトが削除されることなくカラムがインライン・デフォルト・オブジェクトからバインド解除されます。

インライン・デフォルト・オブジェクトは、カラムに使用されなくなると、削除されます。

制限事項

共有インライン・デフォルトには、次の制限が適用されます。

- 共有インライン・デフォルトは、グローバルまたはユーザの tempdb では使用できません。
- 変数を使用するインライン・デフォルトは共有できません。
- ユーザ間で共有インライン・デフォルトは使用できません。
- 公式を使用するインライン・デフォルトは共有できません。
- *constant_value* は、定数リテラルでなければなりません。
- syscomments で定義され、複数のロー (255 バイトを超える) が必要なインライン・デフォルトは、共有できません。
- ダウングレードしている Adaptive Server に存在する共有可能なインライン・デフォルトは、前のリリースで create default コマンド実行時に作成されたかのように扱われます。これらのデフォルトは、内部デフォルト名を使用した sp_binddefault、sp_unbindefault、および drop default コマンドを持ち、完全に機能します。

第 15 章

バッチおよびフロー制御言語の使用

Transact-SQL を使用すると、対話的に、またはオペレーティング・システム・ファイルから、一連の SQL 文をバッチとしてグループ化できます。Transact-SQL のフロー制御言語を使用して、プログラミング構造で複数の文を連結することもできます。

「変数」は、値が代入されるエンティティです。この値は、変数を使用するバッチまたはストアド・プロシージャにおいて変更できます。Adaptive Server には、「ローカル変数」と「グローバル変数」の 2 種類の変数があります。ローカル変数は、ユーザが定義する変数ですが、グローバル変数はコマンドの実行中にシステムから提供される、事前に定義された変数です。

概要

Adaptive Server では、バッチとして送信された複数の文を、対話的に、またはファイルから処理できます。バッチまたはバッチ・ファイルは Transact-SQL 文の集合であり、次々に送信され、グループとして実行されます。バッチは、バッチ終了シグナルで終了します。isql ユーティリティでは、バッチ終了シグナルは “go” で、1 行にこの信号だけを入力します。isql の詳細については、『ユーティリティ・ガイド』を参照してください。

このバッチには、2 つの Transact-SQL 文が含まれています。

```
select count(*) from titles
select count(*) from authors
go
```

1 つの SQL 文によってバッチを構成できますが、複数の文でバッチを構成するのが一般的です。バッチは、isql で実行される前にオペレーティング・システム・ファイルに書き込まれることがよくあります。

Transact-SQL では、文の実行フローを制御する、フロー制御言語と呼ばれる特別なキーワードを使用できます。フロー制御言語は、単一文、バッチ、ストアド・プロシージャ、トリガで使用できます。

フロー制御言語を使用しない場合、それぞれの Transact-SQL 文は、記述順に実行されます。相関サブクエリについては、「第 5 章 サブクエリ：他のクエリ内でのクエリの使用」を参照してください。フロー制御言語によって、プログラム作成と同様の手法を使用して、複数の文を連結および関連付けることができます。

たとえば、コマンドの条件付き実行を表す if...else や反復実行を表す while などのフロー制御言語は、Transact-SQL 文の実行を制御し、効果的にします。Transact-SQL のフロー制御言語を使用することによって、標準的な SQL は、かなり高いレベルのプログラム言語になります。

バッチに関連する規則

どの Transact-SQL 文を連結して 1 つのバッチにできるかを管理する規則があります。このバッチの規則は次のとおりです。

- データベース内のオブジェクトを参照する前に、そのデータベースに対して use 文を発行します。次に例を示します。

```
use master
go
select count(*)
from sysdatabases
go
```

- 次のデータベース・コマンドを、1 つのバッチ内で他の文と組み合わせることはできません。
  - create procedure
  - create rule
  - create default
  - create trigger
- 次のデータベース・コマンドは、1 つのバッチ内で他の Transact-SQL 文と組み合わせることができます。
  - create database (ただし、データベースを作成し、その新しいデータベースにオブジェクトを作成してアクセスすることは、1 つのバッチ内ではできません)
  - create table
  - create index
  - create view

- ルールおよびデフォルトは、カラムにはバインドできず、同一のバッチでは使用できません。sp_bindrule と sp_bindefault は、ルールまたはデフォルトを呼び出す insert 文のあるバッチでは使用できません。
- 1 つのバッチ内で、drop でオブジェクトを削除してから、それを参照または再作成することはできません。
- テーブルが既に存在する場合、そのテーブルの存在テストをバッチに組み込んでいたとしても、そのテーブルをバッチで再作成することはできません。

  Adaptive Server は、バッチをコンパイルしてから実行します。コンパイル中、Adaptive Server は、そのバッチが参照するテーブルやビューなどのオブジェクトの、パーミッション・チェックを行いません。パーミッション・チェックが行われるのは、Adaptive Server がバッチを実行するときです。例外として、Adaptive Server が現在のデータベース以外のデータベースにアクセスするときに、チェックを行います。この場合、Adaptive Server は、バッチ内の文を一切実行せず、コンパイル時にエラー・メッセージを表示します。

  バッチに次の文が含まれているとします。

  ```
  select * from taba
  select * from tabb
  select * from tabc
  select * from tabd
  ```

  3 番目の文 (select * from tabc) 以外のすべての文に必要なパーミッションを持っている場合、Adaptive Server は、その 3 番目の文に対してエラー・メッセージを返し、その他の文に対しては結果を返します。

バッチの使用例

この項では、isql ユーティリティのフォーマットを使用するバッチの例を示します。このバッチには、バッチ終了信号 (1 行に単独で入力された “go” という単語) が含まれています。次に、2 つの select 文を含む 1 つのバッチを示します。

```
select count(*) from titles
select count(*) from authors
go

-------------
          18

(1 row affected)
-------------
          23

(1 row affected)
```

テーブルを作成し、同じバッチでそのテーブルを参照できます。次のバッチを使用して、テーブルを作成し、ローを挿入して、そのロー以降のすべてのデータを選択します。

```
create table test
(column1 char(10), column2 int)
insert test
values ("hello", 598)
select * from test
go

(1 row affected)
column1  column2
-------  -------
hello        598

(1 row affected)
```

起動したデータベースに後続の文で参照するオブジェクトがある場合に限り、use 文を他の文と組み合わせてバッチを作成できます。このバッチは、master データベースにあるテーブルから選択し、次に pubs2 データベースをオープンします。バッチは、master データベースを現在のデータベースにして開始します。例では、pubs2 が現在のデータベースです。

```
use master
go
select count(*) from sysdatabases
use pubs2
go

-------------
           6

(1 row affected)
```

同じバッチ内で、削除したオブジェクトを参照または再作成しない場合に限り、drop 文を他の文と組み合わせてバッチを作成できます。次は、drop 文と select 文を組み合わせたバッチの例です。

```
drop table test
select count(*) from titles
go
------------
         18

(1 row affected)
```

バッチに構文エラーがある場合、その中の文は一切実行されません。次の例は、バッチの最後の文に入力ミスがあり、そのバッチを実行しようとしたために表示されたエラー・メッセージです。

```
select count(*) from titles
select count(*) from authors
```

```
slect count(*) from publishers
go

Msg 156, Level 15, State 1:
Line 3:
Incorrect syntax near the keyword 'count'.
```

バッチの規則に反するバッチを実行した場合も、エラー・メッセージが表示されます。次に、無効なバッチの例を示します。

```
create table test
    (column1 char(10), column2 int)
insert test
    values ("hello", 598)
select * from test
create procedure testproc as
    select column1 from test
go

Msg 111, Level 15, State 7:
Line 6:
CREATE PROCEDURE must be the first command in a
query batch.


create default phonedflt as "UNKNOWN"
sp_bindefault phonedflt, "authors.phone"
go

Msg 102, Level 15, State 1:
Procedure 'phonedflt', Line 2:
Incorrect syntax near 'sp_bindefault'.
```

次のバッチは、ユーザが use 文で指定したデータベースで既に作業している場合に機能します。しかし、master などの別のデータベースからバッチを実行した場合は、エラー・メッセージが表示されます。

```
use pubs2
select * from titles
go

Msg 208, Level 16, State 1:
Server 'hq', Line 2:
titles not found. Specify owner.objectname or use sp_help to
check whether the object exists (sp_help may produce lots of
output)

drop table test
create table test
(column1 char(10), column2 int)
go
Msg 2714, Level 16, State 1:
Server 'hq', Line 2:
There is already an object named 'test' in the
database.
```

ファイルとして送信されるバッチ

Transact-SQL 文で構成されている 1 つ以上のバッチを、オペレーティング・システム・ファイルから isql に送信できます。オペレーティング・システム・ファイルには複数のバッチ、つまり、複数の文の集合を含めることができます。各集合は “go” で終わります。

たとえば、1 つのオペレーティング・システム・ファイルに、次のような 3 つのバッチがあるとします。

```
use pubs2
go
select count(*) from titles
select count(*) from authors
go
create table hello
   (column1 char(10), column2 int)
insert hello
   values ("hello", 598)
select * from hello
go
```

このファイルを isql ユーティリティに送信すると、次の結果が返されます。

```
-------------
           18

(1 row affected)
-------------
           23

(1 row affected)
column1      column2
---------    ---------
hello              598

(1 row affected)
```

ファイルに格納されたバッチの実行に関する環境固有の情報については、『ユーティリティ・ガイド』の「isql」の項を参照してください。

フロー制御言語の使用

フロー制御言語は、対話型の文、バッチ内、ストアド・プロシージャで使用できます。表 15-1 は、フロー制御と関連キーワード、およびその機能のリストです。

表 15-1: フロー制御と関連するキーワード

キーワード	関数
if	条件付き実行を定義する。
…else	if 条件が偽 (false) である場合の代わりの実行を定義する。
case	if...else の代わりに when...then 文を使用して条件式を定義する。
begin	文ブロックを開始する。
...end	文ブロックを終了する。
while	While 条件が真 (true) である間は文を繰り返し実行する。
break	while ループの次の一番外側の終了 (END) から抜ける。
…continue	while ループを再開始する。
declare	ローカル変数を宣言する。
goto label	label: (文ブロック内の指定した場所) へ移動する。
return	実行を無条件に終了する。
waitfor	コマンドの遅延実行を設定する。
print	ユーザ定義メッセージまたはローカル変数を画面に表示する。
raiserror	ユーザ定義メッセージやローカル変数を画面に表示し、グローバル変数 *@@error* 内にシステム・フラグを設定する。
/* *comment* */ または --*comment*	Transact-SQL 文の中の任意の場所にコメントを挿入する。

if...else

if キーワードを使用すると、else の有無に関係なく、次の文を実行するかどうかを決定する条件を設定できます。条件が満たされる (つまり TRUE が返される) と、Transact-SQL 文が実行されます。

else キーワードでは、if で設定した条件が FALSE である場合に実行する代わりの Transact-SQL 文を設定できます。

if と else の構文は、次のとおりです。

```
if
        boolean_expression
        statement
    [else
        [if boolean_expression]
         statement ]
```

ブール式は、TRUE または FALSE を返します。ブール式には、カラム名、定数、算術演算子、またはビット処理演算子で連結したカラム名と定数の組み合わせ、または、サブクエリが 1 つの値を返す場合にはサブクエリを含むことができます。ブール式に select 文を含める場合は、その文をカッコで囲みます。select 文は、1 つの値を返すものでなければなりません。

次に、if を単独で使用した例を示します。

```
if exists (select postalcode from authors
```

```
            where postalcode = "94705")
    print "Berkeley author"
```

authors テーブルにある 1 つ以上の郵便番号の値が “94705” の場合は、メッセージ “Berkeley author” が出力されます。前記の例の select 文は、exists キーワードを条件として設定しているため、TRUE または FALSE のどちらか 1 つの値を返します。この例における exists キーワードは、サブクエリと同様に機能します。「第 5 章 サブクエリ：他のクエリ内でのクエリの使用」を参照してください。

次に、if と else の両方を使用した例を示します。この例では、50 より大きい ID 番号を持つユーザ・オブジェクトの存在をテストします。該当するユーザ・オブジェクトが存在する場合は、else 句がオブジェクトの名前、種類、ID 番号を選択します。

```
if (select max(id) from sysobjects) < 50
   print "There are no user-created objects in this database."
else
   select name, type, id from sysobjects
   where id > 50 and type = "U"

(0 rows affected)

 name           type   id
------------    ----   ----------
 authors        U       16003088
 publishers     U       48003202
 roysched       U       80003316
 sales          U      112003430
 salesdetail    U      144003544
 titleauthor    U      176003658
 titles         U      208003772
 stores         U      240003886
 discounts      U      272004000
 au_pix         U      304004114
 blurbs         U      336004228
 friends_etc    U      704005539
 test           U      912006280
 hello          U     1056006793

(14 rows affected)
```

if...else 構造は、ストアド・プロシージャで頻繁に使用され、特定のパラメータが存在するかどうかをテストします。

if テストは、別の if 内、または else の後にある if テスト内でネストできます。if テストの式は値を 1 つしか返しません。また、各 if...else の構造には、if...else 用の select 文と else 用の select 文があります。複数の select 文を含めるには、begin...end キーワードを使用します。ネストできる if テストの最大数は、各 if...else の構造に含める select 文 (または、他の言語構造) の複雑さによって変わります。

case expression

case 式によって、多数の Transact-SQL 条件構造を簡略化できます。一連の if 文を使用する代わりに case 式を使用すると、条件が一致した場合に適切な値を返す一連の条件を使用できます。case 式は、ANSI-SQL に準拠しています。

case 式を使用すると、次のことができます。

- クエリの簡略化および効率的なコードの作成
- データベースで使用されているフォーマット (int など) とアプリケーションで使用されているフォーマット (char など) 間のデータの変換
- カラム・リスト内の最初の null 以外の値を返す
- 0 による除算を回避するクエリの作成
- 2 つの値を比較して、値が一致しない場合は最初の値を返し、値が一致する場合は null 値を返す。

case 式には、キーワード case、when、then、coalesce、nullif が含まれます。coalesce と nullif は、case 式の省略形です。『リファレンス・マニュアル：コマンド』を参照してください。

代替表現としての *case* 式の使用

case 式を使用すると、データをユーザにとってわかりやすく表現できます。たとえば、pubs2 データベースには、titles テーブルの contract カラムに、本の契約ステータスを示す 1 または 0 が格納されます。しかし、アプリケーション・コードやユーザとの対話では、“Contract” または “No Contract” を使用して本の契約ステータスを表すことで、わかりやすくなる場合があります。代わりの表現を使用して titles テーブルから契約タイプを選択するには、次のように入力します。

```
select title, "Contract Status" =
    case
        when contract = 1 then "Contract"
        when contract = 0 then "No Contract"
    end
from titles
```

```
title                                   Contract Status
-----                                   ---------------
The Busy Executive’s Database Guide     Contract
Cooking with Computers: Surreptitio     Contract
You Can Combat Computer Stress!         Contract
. . .
The Psychology of Computer Cooking      No Contract
. . .
Fifty Years in Buckingham Palace        Contract
Sushi, Anyone?                          Contract

(18 rows affected)
```

case と 0 による除算

case 式を使用すると、0 による除算を回避する (例外の回避と呼ばれます) クエリを作成できます。

この例では、各本の total_sales カラムを advance カラムで除算します。クエリが title_id MC2222 の total_sales (2032) を advance (0.00) で割るときに、そのクエリは 0 による除算となります。

```
select title_id, total_sales, advance, total_sales/advance
from titles

title_id   total_sales          advance
-------    -----------          ---------     ------
BU1032     4095                  5,000.00     0.82
BU1111     3876                  5,000.00     0.78
BU2075     18722                10,125.00     1.85
BU7832     4095                  5,000.00     0.82

Divide by zero occurred.
```

case 式を使用すると、処理する式の中に 0 が含まれないようにして、0 による除算を回避できます。次の例のクエリは、0 を検出したとき、除算を実行する代わりに事前に定義された値を返します。

```
select title_id, total_sales, advance, "Cost Per Book" =
     case
           when advance != 0
           then convert(char, total_sales/advance)
           else "No Books Sold"
     end
from titles

title_id       total_sales     advance          Cost Per Book
--------       -----------     ----------       -------------
BU1032               4095       5,000.00               0.82
BU1111               3876       5,000.00               0.78
BU2075              18722      10,125.00               1.85
BU7832               4095       5,000.00               0.82
MC2222               2032           0.00       No Books Sold
MC3021              22246      15,000.00               1.48
MC3026               NULL           NULL       No Books Sold
. . .
TC3218                375       7,000.00               0.05
TC4203              15096       4,000.00               3.77
TC7777               4095       8,000.00               0.51

(18 rows affected)
```

これで、title_id MC2222 に対する 0 による除算は、クエリの実行を妨げません。また、MC3021 に null 値が入ったとしても、クエリは実行できます。

注意：Adaptive Server は case ロジックを実行する前に定数式を計算するため、除算する数が定数式を計算する場合、case は 0 による除算を回避しません。このためゼロによる除算が発生することがあります。対処するには次のようにします。

- nullif() を使用する。次に例を示します。

  ```
  (x/nullif(@foo.0)0
  ```

- 互いをキャンセルするようにカラム値を含め、Adaptive Server に各行の式を強制的に計算させる。次に例を示します。

  ```
  (x/(@foo + (col1 - col1))
  ```

case 式での rand 関数の使用

rand 関数や getdate 関数などを参照する式は、評価されるたびに異なる値を生成します。このため、特定の case 式でこれらの式を使用すると、予期しない結果が生成されることがあります。たとえば、次の形式の case 式があるとします。

```
case expression
    when value1 then result1
    when value2 then result2
    when value3 then result3
...
end
```

SQL 規格は、この式は次の形式の case 式と等価であると規定しています。

```
case expression
    when expression=value1 then result1
    when expression=value2 then result2
    when expression=value3 then result3
...
end
```

この定義は、検査される各 when 句で式が繰り返し評価されることを明示的に要求しています。case 式の定義は、rand 関数などの関数を参照する case 式に影響します。たとえば、次の case 式があるとします。

```
select
CASE convert(int, (RAND() * 3))
        when 0 then "A"
        when 1 then "B"
        when 2 then "C"
        when 3 then "D"
    else "E"
end
```

SQL 規格によると、この式は次の式と等価であると定義されます。

```
 select
CASE
        when convert(int, (RAND() * 3)) = 0 then "A"
        when convert(int, (RAND() * 3)) = 1 then "B"
        when convert(int, (RAND() * 3)) = 2 then "C"
        when convert(int, (RAND() * 3)) = 3 then "D"
    else "E"
end
```

この形式では、それぞれの when 句で新しい rand 値が生成され、case 式は結果 “E” を頻繁に生成します。

case 式の結果

case 式のデータ型を判別するときの規則は、union 演算でカラムのデータ型を判別する場合の規則に基づいています。case 式には、then 句と else 句で指定する、一連の代替結果式 (下記の例の *R1*、*R2*、...、*Rn*) があります。次に例を示します。

```
case
      when search_condition1 then R1
      when search_condition2 then R2
      ...
      else Rn
   end
```

結果式 *R1*、*R2*、...、*Rn* のデータ型は、case 全体のデータ型の判別に使用されます。*n* 個のテーブルを指定し、*i* 番目のカラムとして *R1*、*R2*、...、*Rn* の式を持つ union のカラムのデータ型を判別するときに使用される規則と同じ規則によって case 式のデータ型も決まります。case のデータ型は、次のクエリと同じ方法で判別されます。

```
select...R1...from ...
union
select...R2...from...
union...
...
select...Rn...from...
```

すべてのデータ型に互換性があるわけではありません。また、互換性がない 2 つのデータ型 (たとえば *char* と *int*) を指定した場合、Transact-SQL クエリは失敗します。『リファレンス・マニュアル：ビルディング・ブロック』を参照してください。

case 式と set ansinull

set ansinull を有効にし、以下に類似の構造のクエリに when null 句をインクルードすると、case 式は null 値に対して異なる結果を出す場合があります。

```
select CVT =
   case case_expression
```

```
      when NULL  then 'YES'
      else 'NO'
   end
from A
```

ansinull を off に設定すると (デフォルト)、これらの case 式は null 値に一致し、述部は null 値を true として評価します (null 値をインクルードした where 句をクエリ・プロセッサが評価する方法と類似)。ansinull を on に設定すると、case 式は null 値を比較せずに、述部を false として評価します。次に例を示します。

```
select CVT =
   case advance
          when  NULL  then 'YES'
          else  'NO'
   end,
advance from titles
--- ------------------------
NO                  5,000.00
NO                 15,000.00
YES                     NULL
NO                  7,000.00
NO                  8,000.00
YES                     NULL
NO                   7,000.00
```

しかし、set ansinull を有効にして同じクエリを実行すると、case 式は null 値に遭遇すると no 値を返します。

```
set ansinull on

select CVT =
   case advance
          when  NULL  then 'YES'
          else  'NO'
   end,
advance from titles
  CVT advance
--- ------------------------
NO                   5,000.00
NO                  15,000.00
NO                      NULL
NO                   7,000.00
NO                   8,000.00
NO                      NULL
NO                    7,000.00
```

『リファレンス・マニュアル：コマンド』を参照してください。

null 以外の結果を少なくとも 1 つ必要とする case 式

case 式の結果の少なくとも 1 つは、null 以外の値を返す必要があります。次のクエリがあるとします。

```
select price,
    case
        when title_id like "%" then NULL
        when pub_id like "%" then NULL
    end
from titles
```

このクエリでは、次のようなエラー・メッセージが返されます。

```
All result expressions in a CASE expression must not be NULL
```

結果セットを決定する

case 式を使用して、結果セットを判別する条件をテストできます。

構文は次のとおりです。

```
case
      when search_condition1 then result1
      when search_condition2 then result2
      . . .
      when search_conditionn then resultn
      else resultx
   end
```

この例の *search_condition* は論理式で、*result* は式です。

search_condition1 が true の場合、case の値は *result1* です。*search_condition1* が true でない場合、*search_condition2* がチェックされます。*search_condition2* が true の場合は、case の値は *result2* と続いていきます。探索条件のどれもが true でない場合、case 値は *resultx* です。else 句はオプションです。この句を使用しないと、デフォルトは else null です。end は case 式の終了を示します。

書店別の各本の売り上げの合計は、salesdetail テーブルに格納されています。本の売り上げ幅を表示する場合は、各書店での各本の売り上げを追跡できます。

- 売り上げ部数が 1000 未満の本 (売り上げが少ない本)
- 売り上げ部数が 1000～3000 の本 (売り上げが中程度の本)
- 売り上げ部数が 3000 より多い本 (売り上げが多い本)

検索するには、次のようなクエリを作成します。

```
select stor_id, title_id, qty, "Book Sales Catagory" =
     case
           when qty < 1000
              then "Low Sales Book"
           when qty >= 1000 and qty <= 3000
              then "Medium Sales Book"
           when qty > 3000
           then "High Sales Book"
     end
from salesdetail
group by title_id
```

```
stor_id     title_id      qty      Book Sales Catagory
-------     --------      ----     -------------------
5023        BU1032         200     Low Sales Book
5023        BU1032        1000     Low Sales Book
7131        BU1032         200     Low Sales Book
. . .
7896        TC7777          75     Low Sales Book
7131        TC7777          80     Low Sales Book
5023        TC7777        1000     Low Sales Book
7066        TC7777         350     Low Sales Book
5023        TC7777        1500     Medium Sales Book
5023        TC7777        1090     Medium Sales Book

(116 rows affected)
```

次の例では、作家の印税率 (*royaltyer*) に従って titleauthor テーブルから titles を選択し、次に各タイトルに High royalty、Medium royalty、または Low royalty という値を代入します。

```
select title, royaltyper, "Royalty Category" =
   case
      when (select avg(royaltyper) from titleauthor tta
        where t.title_id = tta.title_id) > 60 then "High Royalty"
      when (select avg(royaltyper) from titleauthor tta
        where t.title_id = tta.title_id) between 41 and 59
      then "Medium Royalty"
      else "Low Royalty"
   end
from titles t, titleauthor ta
where ta.title_id = t.title_id
order by title
```

```
title                                 royaltyper   royalty Category
-------                               ----------   ----------------
But Is It User Friendly?              100          High Royalty
Computer Phobic and Non-Phobic Ind    25           Medium Royalty
Computer Phobic and Non-Phobic Ind    75           Medium Royalty
Cooking with Computers: Surreptiti    40           Medium Royalty
Cooking with Computers: Surreptiti    60           Medium Royalty
Emotional Security: A New Algorith    100          High Royalty
. . .
Sushi, Anyone?                        40           Low Royalty
The Busy Executive’s Database Guide   40           Medium Royalty
The Busy Executive’s Database Guide   60           Medium Royalty
The Gourmet Microwave                 75           Medium Royalty
You Can Combat Computer Stress!       100          High Royalty

(25 rows affected)
```

case と値の比較

次の形式の case は、値の比較に使用されます。可能なのは、2 つの値が等しいかどうかのチェックだけであり、それ以外の比較はできません。

構文は次のとおりです。

```
case valueT
        when value1 then result1
        when value2 then result2
        . . .
        when valuen then resultn
        else resultx
    end
```

value と *result* は式です。

valueT が *value1* と等しい場合、case の値は *result1* です。*valuetT* が value1 と等しくない場合、*valueT* は *value2* と比較されます。*valueT* が *value2* と等しい場合、case の値は *result2* と続いていきます。*valueT* が *value1* ～ *valuen* までの値と等しくない場合、case の値は *resultx* です。

少なくとも 1 つの結果が null 以外である必要があります。結果式はすべて互換性がなければなりません。また、values もすべて互換性を保つようにする必要があります。

上記の構文は、次の構文と同じことを表しています。

```
case
        when valueT = value1 then result1
        when valueT = value2 then result2
        . . .
        when valueT = valuen then resultn
        else resultx
    end
```

このフォーマットは、case と探索条件で使用されるフォーマットと同じです (「結果セットを決定する」(460 ページ) を参照)。

次の例では、titles テーブルから title と pub_id を選択し、pub_id に基づいて、各本の出版社を指定します。

```
select title, pub_id, "Publisher" =
   case pub_id
     when "0736" then "New Age Books"
     when "0877" then "Binnet & Hardley"
     when "1389" then "Algodata Infosystems"
     else "Other Publisher"
   end
from titles
order by pub_id
```

```
title                          pub_id       Publisher
-----                          ------       -------------
Life Without Fear                0736       New Age Books
Is Anger the Enemy?              0736       New Age Books
You Can Combat Computer          0736       New Age Books
. . .
```

```
Straight Talk About Computers    1389       Algodata Infosystems
The Busy Executive's Database    1389       Algodata Infosystems
Cooking with Computers: Surre    1389       Algodata Infosystems

(18 rows affected)
```

上記のクエリは、次のクエリと等価です。次のクエリは、case と探索条件の構文を使用しています。

```
select title, pub_id, "Publisher" =
   case
     when pub_id = "0736" then "New Age Books"
     when pub_id = "0877" then "Binnet & Hardley"
     when pub_id = "1389" then "Algodata Infosystems"
     else "Other Publisher"
   end
from titles
order by pub_id
```

coalesce

coalesce は、一連の値 (*value1*、*value2*、...、*valuen*) を検査し、最初の null 以外の値を返します。coalesce の構文は次のとおりです。

coalesce(*value1*, *value2*, ..., *valuen*)

value1、*value2*、...、*valuen* は式です。*value1* が null 以外の場合、coalesce の値は *value1* です。*value1* が null の場合、*value2* を検査し、以下同様に処理します。このようにして、null 以外の値が検出されるまで、値の検査が続きます。最初の null 以外の値が coalesce の値になります。

coalesce を使用すると、Adaptive Server は、これを内部で次のように変換します。

```
case
    when value1 is not NULL then value1
    when value2 is not NULL then value2
    . . .
    when valuen-1 is not NULL then valuen-1
    else valuen
end
```

valuen-1 は、最後の値 *valuen* の直前の値を示します。

次の例は、coalesce を使用して、書店の注文が少ない (101 以上 1000 未満) か、多い (1001 以上) かを判別します。

```
select stor_id, discount, "Quantity" =
    coalesce (lowqty, highqty)
from discounts

stor_id discount    Quantity
------- --------   ---------   ----
NULL               10.500000   NULL
NULL                6.700000    100
```

```
NULL                10.000000   1001
8042                 5.000000   NULL

(4 rows affected)
```

nullif

nullif を使用して、コード化された形式で格納されている情報のうち、欠落している情報、認識できない情報、適用できない情報を検索します。たとえば、認識できない値は、履歴上 -1 として格納される場合があります。nullif を使用すると、-1 を null で置き換え、Transact-SQL で定義される null 動作を実行できます。構文は次のとおりです。

nullif(*value1*, *value2*)

value1 が *value2* と等しい場合、nullif は null を返します。*value1* が *value2* と等しくない場合、nullif は *value1* を返します。*value1* と *value2* は式で、これらのデータ型は比較可能でなければなりません。

nullif を使用すると、Adaptive Server は、これを内部で次のように変換します。

```
case
      when value1 = value2 then NULL
      else value1
   end
```

たとえば、titles テーブルは、タイプ・カテゴリがまだ決まっていない本を、値 “UNDECIDED” を使用して表します。次のクエリは、本のタイプについて、titles テーブルを検索します。“UNDECIDED” タイプの本は、タイプ null として返されます (次の出力は、表示用に再フォーマットしたものです)。

```
select title, "type"=
   nullif(type, "UNDECIDED")
from titles

title                                   type
-----                                   --------
The Busy Executive’s Database Guide     business
Cooking with Computers: Surreptiti      business
You Can Combat Computer Stress!
. . .
The Psychology of Computer Cooking      NULL
Fifty Years in Buckingham Palace K      trad_cook
0877                          trad_cook

(18 rows affected)
```

『The Psychology of Computing』は “UNDECIDED” としてテーブルに格納されていますが、クエリはこれをタイプ null として返します。

begin...end

キーワードの begin と end は、一連の文を囲むことができます。囲んだ文は、フロー制御構造によって、if...else のようなユニットとして扱われます。この begin と end で囲まれた一連の文は「文ブロック」と呼ばれます。

次に、begin...end の構文を示します。

```
begin
        statement block
end
```

次に例を示します。

```
if (select avg(price) from titles) < $15
begin
   update titles
   set price = price * 2

   select title, price
   from titles
   where price > $28
end
```

begin と end を使用しない場合、if 条件は最初の Transact-SQL 文にだけ適用されます。2 番目の文は、最初の文とは関係なく実行されます。

begin...end の構文で囲まれた文ブロックは、ほかの begin...end の文ブロック内でネストできます。

while と break...continue

while キーワードを使用して、文または文ブロックの反復実行の条件を設定できます。設定した条件が true である場合には、文は反復実行されます。

構文は次のとおりです。

```
while boolean_expression
    statement
```

次の例では、平均価格が 30 ドル未満である限り、select 文と update 文が反復実行されます。

```
while (select avg(price) from titles) < $30
begin
   select title_id, price
   from titles
   where price > $20
   update titles
   set price = price * 2
end

(0 rows affected)

title_id    price
```

```
------      -------
PC1035        22.95
PS1372        21.59
TC3218        20.95

(3 rows affected)
(18 rows affected)
(0 rows affected)
title_id    price
------      -------
BU1032      39.98
BU1111      23.90
BU7832      39.98
MC2222      39.98
PC1035        45.90
PC8888      40.00
PS1372        43.18
PS2091      21.90
PS3333      39.98
TC3218        41.90
TC4203      23.90
TC7777      29.98

(12 rows affected)
(18 rows affected)
(0 rows affected)
```

break と continue は、while ループ内の文の実行を制御します。break を使用すると、while ループを終了します。ループの終了を表す end キーワードの後にある文は、引き続き実行されます。continue によって、ループ内以外の continue の後の文をスキップして、while ループを再実行できます。break と continue は、if テストによってアクティブにされることがよくあります。

次に、break...continue の構文を示します。

```
while boolean expression
    begin
        statement
        [statement]...
          break
        [statement]...
        continue
        [statement]...
    end
```

次は、前の例で行われた値上げに対して、while、break、continue、if を使用して値下げする例です。平均価格が 20 ドルを超えるときは、価格はすべて半分になります。次に、最高値が選択されます。最高値が 40 ドル未満であれば、while ループは終了します。最高値が 40 ドル以上であれば、もう一度ループを回します。平均値が 20 ドルを超える場合に限り、continue は print 文の実行を許可します。while ループが終了すると、メッセージと最も値段の高い本のリストが表示されます。

```
while (select avg(price) from titles) > $20
begin
    update titles
        set price = price / 2
    if (select max(price) from titles) < $40
        break
    else
        if (select avg(price) from titles) < $20
            continue
    print "Average price still over $20"
end

select title_id, price from titles
    where price > $20

print "Not Too Expensive"
```

```
(18 rows affected)
(0 rows affected)
(0 rows affected)
Average price still over $20
(0 rows affected)
(18 rows affected)
(0 rows affected)

title_id    price
--------    -------
PC1035        22.95
PS1372        21.59
TC3218        20.95

(3 rows affected)
Not Too Expensive
```

2 つ以上の while ループがネストされている場合は、break によって現在のループから 1 つ外側のループに移ることができます。最初に内側のループの end の後のすべての文が実行されます。次に、外側のループが再開始されます。

declare とローカル変数

ローカル変数は、declare キーワードによって宣言、命名、入力し、select 文によって初期値を代入します。これは同一のバッチまたはプロシージャ内で行う必要があります。

「ローカル変数」(475 ページ) を参照してください。

goto

goto キーワードによって、ユーザ定義ラベルに無条件分岐が発生します。goto とラベルは、ストアド・プロシージャおよびバッチで使用できます。ラベルの名前は、識別子の規則に従う必要があります。ラベル名を最初に指定するときは、ラベル名の後にコロンを付けます。goto とともに使用される場合は、コロンは付けません。

goto の構文は次のとおりです。

label:
goto *label*

この例では、goto、ラベル、while ループ、ローカル変数をカウンタとして使用します。

```
declare @count smallint
select @count = 1
restart:
print "yes"
select @count = @count + 1
while @count <=4
   goto restart
```

goto は通常、while か if テスト、または他の状態に依存して、goto とラベルの間で無限ループが発生しないようにします。

return

キーワード return によって、バッチまたはプロシージャから無条件に抜けることができます。このコマンドは、バッチまたはプロシージャのどこででも使用できます。ストアド・プロシージャで使用する場合、return にはオプションの引数を指定でき、ステータスを呼び出し側に返します。return の後の文は実行されません。

構文は次のとおりです。

return [*int_expression*]

次に、return、if...else、begin...end を使用したストアド・プロシージャの例を示します。

```
create procedure findrules @nm varchar(30) = null as
if @nm is null
begin
  print "You must give a user name"
return
end
else
begin
   select sysobjects.name, sysobjects.id, sysobjects.uid
     from sysobjects, master..syslogins
     where  master..syslogins.name = @nm
```

```
        and sysobjects.uid = master..syslogins.suid
        and sysobjects.type = "R"
end
```

findrules が呼び出されたときに、ユーザ名がパラメータとして与えられていない場合、return キーワードによって、メッセージがユーザ画面に表示された後にプロシージャを終了することができます。ユーザ名が与えられている場合は、ユーザが所有しているルール名が適切なシステム・テーブルから検索されます。

return は、while ループで使用されるキーワード break に似ています。

戻り値の使用例については、「第 17 章 ストアド・プロシージャの使用」を参照してください。

print

前の例で使用されたキーワード print によって、ユーザ定義メッセージまたはローカル変数の内容が画面に表示されます。ローカル変数は、使用される同一のバッチまたはプロシージャ内で宣言します。メッセージの長さは、最大 255 バイトです。

構文は次のとおりです。

```
print {format_string | @local_variable |
      @@global_variable} [,arg_list]
```

次に例を示します。

```
if exists (select postalcode from authors
   where postalcode = "94705")
print "Berkeley author"
```

次は、ローカル変数の内容を表示させる場合の print の使い方です。

```
declare @msg char(50)
select @msg = "What's up, doc?"
  print @msg
```

print は、出力される文字列のプレースホルダを認識します。フォーマット文字列には、20 個までのユニークなプレースホルダを順不同で含めることができます。プレースホルダは、メッセージのテキストがクライアントに送信されるときに、*format_string* に続く引数のフォーマットされた内容に置き換えられます。

フォーマット文字列が異なる文法構造の言語に翻訳されるときに、引数の再編成ができるように、プレースホルダに番号が付けられます。1 つの引数のプレースホルダは、*%nn!* の形式で表現されます。パーセント記号の後に 1 ～ 20 の整数を指定し、感嘆符を付けます。整数は、元の言語の文字列におけるプレースホルダの位置を表現します。“%1!” は元のバージョンでの最初の引数、“%2!” は 2 番目の引数となります (以降同様)。引数の位置を示すことによって、翻訳された言語で現れる引数の順序が元の言語での順序と異なる場合でも正確に翻訳できます。

たとえば、次は英語で書かれたメッセージです。

```
%1! is not allowed in %2!.
```

次は、このメッセージをドイツ語で書いたものです。

```
%1! ist in %2! nicht zulässig.
```

次は、このメッセージを日本語で書いたものです。

図 15-1: 日本語のメッセージ

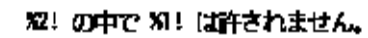


図 15-1 は、日本語のフレーズを示しています。このフレーズでは、異なる場所に文字 “%1!” が含まれます。この例では、英語、ドイツ語、日本語の “%1!” も “%2!” も、3 つの言語で単一の引数を表します。

プレースホルダを番号順に使用する必要はありませんが、フォーマット文字列でプレースホルダを使用するときには、プレースホルダの番号をとばすことはできません。たとえば、フォーマット文字列にプレースホルダ 2 がなくて、プレースホルダ 1 と 3 を持つことはできません。

オプションの *arg_list* は、一連の変数または定数の場合があります。引数は、text または image 以外のすべてのデータ型にすることができ、最終メッセージに含められる前に char データ型に変換されます。引数リストが提供されない場合、フォーマット文字列はプレースホルダを持たず、メッセージとして表示されます。

置き換えられたすべての引数と *format_string* を加えた出力文字列の長さは、最大 512 バイトです。

raiserror

raiserror によって、ユーザ定義のエラーかローカル変数のメッセージが画面に表示され、エラーの発生を記録するシステム・フラグが設定されます。print と同様に、ローカル変数は使用されるバッチまたはプロシージャで宣言します。メッセージの長さは、最大 255 文字です。

raiserror の構文は次のとおりです。

```
raiserror error_number
    [{format_string | @local_variable}] [, arg_list]
    [extended_value  = extended_value  [{,
     extended_value  = extended_value }...]]
```

error_number は、Adaptive Server によって最後に作成されたエラー番号を格納する、グローバル変数 *error@@* に置かれます。ユーザ定義エラー・メッセージ用のエラー番号には、17,000 より大きい数値を使用します。*error_number* が 17,000 ～ 19,999 で、*format_string* がないか、または空 (“ ”) である場合、Adaptive Server は、エラー・メッセージ・テキストの検索を、master データベース内の sysmessages テーブルから実行します。これらエラー・メッセージは、主にシステム・プロシージャによって使用されます。

format_string 自体の長さは、最大 255 バイトに制限されます。*format_string* にすべての引数を加えた出力の長さは、最大で 512 バイトです。raiserror メッセージ用に使用されるローカル変数は、char または varchar です。*format_string* または変数はオプションです。このいずれかを含めない場合、Adaptive Server は、デフォルト言語の sysusermessages にある *error_number* に対応するメッセージを使用します。print と同様に、*arg_list* によって定義された変数または定数を *format_string* に代入できます。

Open Client アプリケーションで使用する拡張されたエラーのデータを定義できます (*extended_values* に raiserror を指定した場合)。拡張されたエラー・データの詳細については、Open Client のマニュアルまたは『リファレンス・マニュアル：コマンド』を参照してください。

error@@ にエラー番号を格納する場合は、print ではなく raiserror を使用します。次の例に、findrules プロシージャにおいて raiserror を使用する方法を示します。

```
raiserror 99999 "You must give a user name"
```

すべてのユーザ定義エラー・メッセージの重大度レベルは 16 です。このレベルは、ユーザが致命的でない間違いをしたことを示します。

print および *raiserror* のためのメッセージ作成

sp_getmessage とともに、print または raiserror を使用して、sysusermessages からメッセージを呼び出すことができます。メッセージ・セットを作成するには、sp_addmessage を使用します。

次の例は、sp_addmessage、sp_getmessage、print を使用して、メッセージを sysusermessages 内に英語とドイツ語の両方でインストールし、そのメッセージをユーザ定義のストアド・プロシージャで使用するために取り出して出力します。

```
/*
** Install messages
** First, the English (langid = NULL)
*/
set language us_english
go
sp_addmessage 25001,
  "There is already a remote user named '%1!' for remote server
'%2!'."
go
```

```
/* Then German*/
sp_addmessage 25001,
        "Remotebenutzername '%1!' existiert bereits auf dem
Remoteserver '%2!'.","german"
go
create procedure test_proc @remotename varchar(30),
            @remoteserver varchar(30)
as
      declare @msg varchar(255)
        declare @arg1 varchar(40)
        /*
        ** check to make sure that there is not
        ** a @remotename for the @remoteserver.
        */
        if exists (select *
            from master.dbo.sysremotelogins l,
                master.dbo.sysservers s
            where l.remoteserverid = s.srvid
                and s.srvname = @remoteserver
                and l.remoteusername = @remotename)
      begin
            exec sp_getmessage 25001, @msg output
            select @arg1=isnull(@remotename, "null")
            print @msg, @arg1, @remoteserver
            return(1)
      end
return(0)
go
```

また、「制約のエラー・メッセージの作成」(285 ページ) で説明されているように、ユーザ定義メッセージを制約にバインドすることもできます。

ユーザ定義メッセージを削除するには、sp_dropmessage を使用します。メッセージを変更するには、sp_dropmessage でメッセージを削除してから sp_addmessage で再び追加します。

waitfor

waitfor キーワードを使用して、1 日の特定の時刻、時間間隔、または、文ブロック、ストアド・プロシージャ、トランザクションを実行するイベントを指定できます。

構文は次のとおりです。

```
waitfor {delay "time" | time "time" | errorexit |  processexit | mirrorexit}
```

ここで、delay *time* は、Adaptive Server に、指定された長さの時間が経過するまで待機するように指示します。time '*time*' は、Adaptive Server に、datetime データとして有効なフォーマットで指定された時刻まで待機するように指示します。

ただし、日付は指定できません (datetime 値の日付部分は無効です)。waitfor time または waitfor delay によって、時、分、秒を、最大 24 時間まで設定できます。“hh:mm:ss” のフォーマットを使用してください。

たとえば、次のコマンドは、Adaptive Server に、午後 4 時 23 分まで待機するように指示します。

```
waitfor time "16:23"
```

次のコマンドは、Adaptive Server に、1 時間 30 分待機するように指示します。

```
waitfor delay "01:30:00"
```

time 値のフォーマットについては、「時刻の入力」(209 ページ) を参照してください。

errorexit は、Adaptive Server に、プロセスの異常終了が発生するまで待機するように指示します。processexit は、何らかの理由でプロセスが終了するまで待機します。mirrorexit は、ミラーリングされたデバイスに対する読み込みまたは書き込みが失敗するまで待機します。

waitfor errorexit を、異常終了したプロセスを強制終了するプロシージャで使用することにより、異常になったプロセスに占有されたシステム・リソースを解放できます。異常なプロセスを検出するには、sp_who を使用して sysprocesses テーブルをチェックします。

次の例は、Adaptive Server に対し、午後 2 時 20 分まで待機するように指示します。そして、指定の時間になると、chess テーブルを新しいプロセスで更新し、sendmessage というストアド・プロシージャを実行します。これは、chess テーブルに新しいプロセスが発生したことを通知するメッセージを、Judy のテーブルのうちの 1 つに挿入します。

```
begin
waitfor time "14:20"
insert chess(next_move)
values("Q-KR5")
execute sendmessage "Judy"
end
```

午後 2 時 20 分まで待機するのではなく、10 秒後にメッセージを Judy に送信するには、上の例の waitfor 文を次のように変更します。

```
waitfor delay "0:00:10"
```

waitfor コマンドを実行したあとは、設定した時刻またはイベントの発生まで、Adaptive Server への接続は使用できません。

コメント

コメント表記法に従い、文、バッチ、ストアド・プロシージャにコメントを付加できます。コメントは実行されません。またコメントの長さに制限はありません。

コメントは、単独行に挿入したり、コマンド・ラインの末尾に挿入できます。コメントの表記の仕方は 2 種類あります。1 つは次に示す、スラッシュとアスタリスクを使用した形式です。

```
/* text of comment */
```

もう 1 つは次の「二重ハイフン」の形式です。

```
-- text of comment
```

スラッシュとアスタリスク形式のコメント

/* 形式のコメントは、Transact-SQL 拡張機能です。複数行のコメントも、各コメントが “/*” で始まり、“*/” で終わるようにすれば入力できます。“/*” と “*/” の間にあるすべての文字は、コメントの一部として扱われます。/* によるフォームはネストできます。

複数行のコメントによく使用される表記規則では、最初の行を “/*” で開始し、以降の行を “**” で開始します。コメントは、通常どおり “*/” で終了します。

```
select * from titles
/* A comment here might explain the rules
** associated with using an asterisk as
** shorthand in the select list.*/
where price > $5
```

次の例は、複数のコメントを含むプロシージャです。

```
/* this procedure finds rules by user name*/
create procedure findmyrule @nm varchar(30) = null
as
if @nm is null
begin
  print "You must give a user name"
return
  print "I have returned"
/* this statement follows return,
** so won't be executed */
end
else      /* print the rule names and IDs, and
          the user ID */
  select sysobjects.name, sysobjects.id,
     sysobjects.uid
  from sysobjects, master..syslogins
  where  master..syslogins.name = @nm
  and sysobjects.uid = master..syslogins.suid
  and sysobjects.type = "R"
```

二重ハイフン形式のコメント

このコメント形式は、2 つの連続したハイフンで始まり、その後にスペースが 1 つ入り (--)、復帰改行文字で終わります。そのため、複数行にわたるコメントは入力できません。

Adaptive Server は、文字列リテラルまたは /* 形式のコメント内の 2 つの連続したハイフンを、コメントの開始とは解釈しません。

2 つの連続したマイナス記号 (後に単項が続くバイナリ) を含む式を表すには、2 つのハイフンの間にスペースを 1 つ挿入するか、左カッコを 1 つ挿入します。

次にそのようなイベントの例を示します。

```
-- this procedure finds rules by user name
create procedure findmyrule @nm varchar(30) = null
as
if @nm is null
begin
  print "You must give a user name"
return
  print "I have returned"
-- each line of a multiple-line comment
-- must be marked separately.
end
else        -- print the rule names and IDs, and
        -- the user ID
  select sysobjects.name, sysobjects.id,
     sysobjects.uid
  from sysobjects, master..syslogins
  where  master..syslogins.name = @nm
  and sysobjects.uid = master..syslogins.suid
  and sysobjects.type = "R"
```

ローカル変数

ローカル変数は、while ループや if...else ブロックのカウンタとして、バッチやストアド・プロシージャでよく使用されます。ストアド・プロシージャで使用されている場合、そのプロシージャの実行時にそれらのローカル変数が自動的かつ非対話的に使用されることが宣言されます。*char_expr*、*integer_expression*、*numeric_expr*、*float_expr* などのように、Transact-SQL 構文で式が使用可能と示されているほとんどの場所で、変数を使用できます。

ローカル変数の宣言

declare を使用して、ローカル変数の名前とデータ型を宣言するには、次の構文を使用します。

declare @*variable_name datatype*
  [, @*variable_name datatype*]...

変数名は、@ 記号で始め、識別子の規則に従って指定します。text、image、sysname 以外のユーザ定義データ型かシステムから提供されるデータ型を指定します。

メモリとパフォーマンスの点からは、次のように指定すると効率的です。

```
declare @a int, @b char(20), @c float
```

これはあまり効率的でない例です。

```
declare @a int
declare @b char(20)
declare @c float
```

ローカル変数と *select* 文

変数を宣言すると、その変数には null 値が代入されます。ローカル変数に値を代入するには、select 文を使用します。

declare 文と同様に、次のように指定すると効率的です。

```
select @a = 1, @b = 2, @c = 3
```

これはあまり効率的でない例です。

```
select @a = 1
select @b = 2
select @c = 3
```

『リファレンス・マニュアル：コマンド』を参照してください。

単一の select 文の中で、ある変数に値を代入してから、さらに、別の変数にその最初の変数に基づく値を代入しないでください。これを行うと、期待どおりの結果が得られない場合があります。たとえば、次の 2 つのクエリは、両方とも *@c2* の値を取り出そうとしています。最初のクエリは null を返し、2 番目のクエリは正しい答え 0.033333 を返します。

```
/* this is wrong*/
declare @c1 float, @c2 float
select @c1 = 1000/1000, @c2 = @c1/30
select @c1, @c2

/* do it this way */
declare @c1 float, @c2 float
select @c1 = 1000/1000
select @c2 = @c1/30
select @c1, @c2
```

変数に値を代入する select 文を使用して、ユーザにデータを返すことはできません。次の例の最初の select 文は、ローカル変数 *@veryhigh* に最高価格を代入します。2 番目の select 文は、値を表示するために必要です。

```
declare @veryhigh money
select @veryhigh = max(price)
   from titles
select @veryhigh
```

変数に値を代入する select 文が複数の値を返す場合は、返される最後の値が変数に代入されます。次のクエリは、“select advance from titles” によって返される最後の値を変数に代入します。

```
declare @m money
select @m = advance from titles
select @m

(18 rows affected)
------------------------
                8,000.00

(1 row affected)
```

代入文は、select 文によって影響を受けた (返された) ローの数を示します。

変数に値を代入する select 文が値を返せないと、変数はその文によって変更されません。

ローカル変数は、print または raiserror の引数として使用できます。

update 文で変数を使用する場合は、「update での set 句の使用」(232 ページ)を参照してください。

ローカル変数と *update* 文

update 文の中で変数への代入を直接行うことができます。select 文を使用して変数に値を代入する必要はありません。変数を宣言したときは、その変数は値 null を持っています。

『リファレンス・マニュアル：コマンド』を参照してください。

ローカル変数とサブクエリ

ローカル変数に値を代入するサブクエリは、値を 1 つだけ返すことができます。例を示します。

```
declare @veryhigh money
select @veryhigh = max(price)
   from titles
if @veryhigh > $20
   print "Ouch!"
```

```
declare @one varchar(18), @two varchar(18)
select @one = "this is one", @two = "this is two"
if @one = "this is one"
   print "you got one"
if @two = "this is two"
   print "you got two"
else print "nope"
declare @tcount int, @pcount int
select @tcount = (select count(*) from titles),
   @pcount = (select count(*) from publishers)
select @tcount, @pcount
```

ローカル変数、*while* ループ、*if...else* ブロック

次の例は、while ループのカウンタにローカル変数を使用して、if 文内の where 句でマッチングを行い、select 文中の値の設定やリセットを行います。

```
/* Determine if a given au_id has a row in au_pix*/
/* Turn off result counting */
set nocount on
/* declare the variables */
declare @c int,
        @min_id varchar(30)
/*First, count the rows*/
select @c = count(*) from authors
/* Initialize @min_id to "" */
select @min_id = ""
/* while loop executes once for each authors row */
while @c > 0
begin
        /*Find the smallest au_id*/
        select @min_id = min(au_id)
            from authors
            where au_id > @min_id
        /*Is there a match in au_pix?*/
        if exists (select au_id
            from au_pix
            where au_id = @min_id)
            begin
            print "A Match!%1!", @min_id
      end
      select @c = @c -1  /*decrement the counter */
end
```

変数と null 値

ローカル変数は、宣言されたときに値 null が代入されます。また、select 文によって null 値が代入される場合もあります。null に特別な意味をもたせるためには、変数に設定された null 値と他の null 値を比較するときに、特別な規則を適用することが必要です。

表 15-2 は、異なる比較演算子を使用する null 値カラムと null 値式の比較の結果を示しています。式は、変数、リテラル、または変数、リテラル、算術演算子の組み合わせで表せます。

表 15-2: null 値の比較

比較のタイプ	= 演算子の使用	<、>、<=、!=、!<、!>、または <> 演算子の使用
column_value と *column_value* の比較	FALSE	FALSE
column_value と *expression* の比較	TRUE	FALSE
expression と *column_value* の比較	TRUE	FALSE
expression と *expression* の比較	TRUE	FALSE

次にテストの例を示します。

```
declare @v int, @i int
if @v = @i select "null = null, true"
if @v > @i select "null > null, true"
```

このテストでは、最初の比較だけが true を返します。

```
-----------------
null = null, true

(1 row affected)
```

次の例は、titles テーブルから、advance に null 値があるローをすべて返します。

```
declare @m money
select title_id, advance
from titles
where advance = @m
title_id  advance
-------- ----------------
MC3026              NULL
PC9999              NULL

(2 rows affected)
```

グローバル変数

グローバル変数は、システムによって提供される事前定義済みの変数で、Transact-SQL の拡張機能です。グローバル変数は、ローカル変数と区別するために、名前の先頭に @ 記号を 2 つ付けます (たとえば、*@@error*)。2 つの @ 記号は、グローバル変数を定義するための識別子の一部とみなされます。

ユーザは、グローバル変数を作成したり、select 文の中で直接グローバル変数の値を更新したりできません。ユーザが「グローバル変数」と同じ名前のローカル変数を宣言した場合、その変数はローカル変数とみなされます。

グローバル変数の完全なリストについては、『リファレンス・マニュアル：ビルディング・ブロック』の「第 3 章 グローバル変数」を参照してください。

トランザクションとグローバル変数

グローバル変数の中には、トランザクションで使用される情報を提供するものもあります。

@@error によるエラーのチェック

@@error グローバル変数は、通常は、現在のユーザ・セッションで最後に実行されたバッチのエラー・ステータスを確認するために使用されます。最後のトランザクションが成功した場合、*@@error* には 0 が入っています。それ以外の場合、*@@error* にはシステムが生成した最後のエラー番号が入っています。if @@error != 0 という文の後に return 句が続く場合、エラーが発生すると終了します。

print 文や if テストを含む、すべての Transact-SQL 文によって *@@error* はリセットされるので、ステータス・チェックは、成功が未確認のバッチの直後に行う必要があります。

@@sqlstatus グローバル変数は、*@@error* の出力に影響しません。「最後の fetch からのステータスのチェック」(481 ページ) を参照してください。

@@identity による IDENTITY 値のチェック

@@identity には、現在のユーザ・セッションの IDENTITY カラムに挿入された最後の値が含まれています。*@@identity* は、insert、select into、bcp がテーブルにローを挿入しようとするたびに設定されます。insert 文、select into 文、または bcp 文の失敗や、その文を含むトランザクションのロールバックは、*@@identity* の値には影響しません。*@@identity* は、IDENTITY カラムに最後の値を挿入した文がコミットに失敗した場合でも、その値を保持します。

文が複数のローを挿入した場合、*@@identity* は、最後に挿入されたローの IDENTITY 値を反映します。影響を受けたテーブルに IDENTITY カラムがない場合、*@@identity* は 0 に設定されます。

@@*trancount* によるトランザクション・ネスト・レベルのチェック

@@*trancount* には、現在のユーザ・セッションでトランザクションのネスト・レベルが設定されます。バッチ内で begin transaction を検出するたびに、トランザクション・カウントが増えます。連鎖トランザクション・モードで @@*trancount* を問い合わせると、クエリが自動的にトランザクションを開始するので、その値は必ず 0 以外になります。

@@*transtate* によるトランザクション状態のチェック

@@*transtate* には、現在のユーザ・セッションで実行した後のトランザクションの現在の状態が設定されています。@@*error* とは違って、@@*transtate* はバッチごとにクリアされません。表 15-3 に、@@*transtate* に設定される値を示します。

表 15-3: @@transtate 値

値	意味
0	トランザクションが進行中。明示的または暗黙的トランザクションが行われている。
1	トランザクションが成功した。トランザクションが完了し、その変更がコミットされた。
2	文がアボートされた。前の文がアボートされた。トランザクションには影響しない。
3	トランザクションがアボートされた。トランザクションがアボートされ、すべての変更がロールバックされた。

@@*transtate* は、実行エラーによってだけ変更されます。構文エラーとコンパイル・エラーは、@@*transtate* の値に影響しません。トランザクションでの @@*transtate* の使用例については、「トランザクションのステータスの確認」(667 ページ)を参照してください。

@@*nestlevel* によるネスト・レベルのチェック

@@*nestlevel* には、ユーザ・セッションでの現在の実行のネスト・レベルが含まれます。初期値は 0 です。ストアド・プロシージャまたはトリガが、別のストアド・プロシージャやトリガを呼び出すたびに、ネスト・レベルが 1 つずつ増加します。ネスト・レベルは、キャッシュされる文が作成されたときも 1 つ増えます。最大値の 16 を超えると、トランザクションはアボートします。

最後の *fetch* からのステータスのチェック

@@*sqlstatus* には、現在のユーザ・セッションの最後の fetch 文によって取り出されるステータス情報が含まれます。@@*sqlstatus* に含まれる値は、次のとおりです。

表 15-4: @@sqlstatus 値

値	意味
0	fetch 文が正常に終了した。
1	fetch 文がエラーになった。
2	結果セットにこれ以上データがない。この警告は、現在のカーソルが結果セットの最終ローを指しており、クライアントがそのカーソルに対して fetch コマンドを送った場合に表示される。

@@sqlstatus グローバル変数は、*@@error* の出力に影響しません。たとえば、次のバッチでは、fetch*@@error* 文の結果をエラーにすることで、*@@sqlstatus* を 1 に設定します。ただし、*@@error* は、*@@sqlstatus* の出力ではなく、エラー・メッセージ番号を反映します。

```
declare csr1 cursor
for select * from sysmessages
for read only

open csr1

begin
    declare @xyz varchar(255)
    fetch csr1 into @xyz
    select error = @@error
    select sqlstatus = @@sqlstatus
end

Msg 553, Level 16, State 1:
Line 3:
FETCH INTO 句にあるパラメータまたは変数の数が、カーソル 'csr1' の結果
セットのカラム数と一致しません。
```

この時点で、*@@error* グローバル変数は、最後に生成されたエラー番号 553 に設定されます。*@@sqlstatus* は 1 に設定されます。

@@fetch_status は、最新の fetch のステータスを返します。

表 15-5: @@fetch_status

値	意味
0	fetch 操作が成功した。
-1	fetch 操作が失敗した。
-2	今後のために予約済み。

第 16 章

クエリでの Transact-SQL 関数の使用

Adaptive Server の関数は、データベースまたはシステム・テーブルから情報を返す Transact-SQL ルーチンです Adaptive Server は、組み込み関数のセットを提供します。さらに、次を使用できます。create function コマンドを使用して Transact-SQL 関数と SQLJ 関数を作成できます。

組み込み関数の完全なリストや説明については、『リファレンス・マニュアル：ビルディング・ブロック』を参照してください。create function コマンドの詳細については、『リファレンス・マニュアル：コマンド』を参照してください。

クエリの設定

通常、関数は、select リスト、where 句、または式が使用できる任意の箇所で使用できます。一般的な構文を示します。

```
select function_name[(arguments)]
```

たとえば、“emilya” でログインするユーザ ID 番号を探すには、次のように入力します。

```
select user_id("emilya")
```

サーバは次を返します。

```
------------------
              1209
```

組み込み関数

複数のさまざまな種類の組み込み関数があります。これらの関数を使用する方法を管理する規則は、異なる可能性があります。この項では、さまざまな種類の関数について説明し、使用方法の基礎知識を提供します。詳細については、『リファレンス・マニュアル：ビルディング・ブロック』、XML 関数については、『XML サービス』を参照してください。

システム関数

システム関数はデータベースに関する情報を返します。その多くは、システム・テーブルに問い合わせを行う手軽な手段です。システム関数への引数がオプションの場合は、現在のデータベース、ホスト・コンピュータ、サーバ・ユーザ、またはデータベース・ユーザを前提としています。user 以外のシステム関数は、引数が null の場合でも必ずカッコを付けて使用します。

たとえば、現在のユーザのユーザ名を検索するには、引数を省略しますが、カッコを付けます。次に例を示します。

```
select user_name()
-------------------
dbo
```

文字列関数

文字列関数は、文字列、式、また、場合によってはバイナリ・データでのさまざまな演算を実行します。文字列関数内で定数を使用する場合は、定数を一重または二重の引用符で囲んでください。次のようにして、バイナリ式と文字式を連結したり (「式の連結」(485 ページ) を参照)、ネストできます (「ネスト文字列関数」(486 ページ) を参照)。

ほとんどの文字列関数は、char、nchar、unichar、varchar、univarchar、および nvarchar の各データ型と、char、unichar や varchar、univarchar に暗黙的に変換されるデータ型でしか使用できません。また、binary と varbinary データで使用できる文字列関数もいくつかあります。patindex は、text、unitext、char、nchar、unichar、varchar、nvarchar、および univarchar カラムで使用できます。

また、各関数には、指定のデータ型に暗黙的に変換できる引数を指定することもできます。たとえば、概数値の式を受け入れる関数は、整数式も受け入れます。Adaptive Server により、引数は指定のデータ型に自動的に変換されます。

式の連結

バイナリ式または文字式は連結できます。つまり、2 つ以上の文字列かバイナリ文字列、文字データかバイナリ・データ、またはそれらの組み合わせを + 文字列連結演算子と結合できます。連結された文字列の最大長は 16384 バイトです。

binary、varbinary カラムと、char、unichar、nchar、varchar、univarchar、nvarchar カラムを連結できます。unichar と univarchar を char、nchar、nvarchar、varchar と連結すると、unichar または univarchar になります。text、unitext、または image カラムは連結できません。

連結の構文は次のとおりです。

select (*expression* + *expression* [+ *expression*]...)

たとえば、2 つの文字列を結合する例を示します。

```
select ("abc" + "def")

-------
abcdef

(1 row affected)
```

Moniker をカラム見出しとして、カリフォルニア州に住む作家名を、姓、名前の順にカンマとスペースで区切って次に入力します。

```
select Moniker = (au_lname + ", " + au_fname)
from authors
where state = "CA"

Moniker
------------------------------------------------
White, Johnson
Green, Marjorie
Carson, Cheryl
O'Leary, Michael
Straight, Dick
Bennet, Abraham
Dull, Ann
...
```

文字列関数で 2 つの文字式が受け入れられるが、1 つの式が unichar の場合、もう 1 つの式が「拡大」され、内部で unichar に変換されます。これは、混合モードの式に関する既存のルールに従っています。ただし、unichar データが 2 倍のスペースを占有することがあるため、この変換によってトランケーションが発生することがあります。

非文字式または非バイナリのカラムを連結するには、convert 関数を使用します。次に例を示します。

```
select "The due date is " + convert(varchar(30),
    pubdate)
```

```
from titles
where title_id = "BU1032"
--------------------------------------
The due date is Jun 12 2006 12:00AM
```

Adaptive Server では、空文字列 (“ ” または ‘ ’) は、1 つのスペースとして評価されます。次のような文があるとします。

```
select "abc" + "" + "def"
```

結果は、次のようになります。

```
abc def
```

連結演算子と LOB ロケータ

+ および || Transact-SQL 演算子は、LOB ロケータを連結演算用の式として受け入れます。1 つまたは複数のロケータを伴う連結演算の結果は、入力ロケータにより参照されるロケータと同じデータ型の LOB ロケータになります。

たとえば、*@v* および *@w* はテキスト・ロケータの変数であると仮定します。有効な連結演算子を次に示します。

- select @v + @w
- select @v || "abdcef"
- select "xyz" + @w

ネスト文字列関数

文字列関数は、ネストできます。たとえば、substring 関数を使用して、作家ごとに姓と名前の頭文字をカンマで区切り、最後にピリオドを付けて表示するには、次のように入力します。たとえば、次のように入力します。

```
select (au_lname + "," + " " + substring(au_fname, 1, 1) + ".")
from authors
where city = "Oakland"

------------------------------------------
Green, M.
Straight, D.
Stringer, D.
MacFeather, S.
Karsen, L.
```

20 ドルを超える本の pub_id と、title_id の最初の 2 文字を表示するには、次のように入力します。

```
select substring(pub_id + title_id, 1, 6)
from titles
where price > $20
--------------
1389PC
```

```
0877PS
0877TC
```

文字列関数の制限

文字列関数の結果は 16KB に制限されます。この制限はサーバのページ・サイズに依存しません。ページ・サイズが 2KB の場合でも、Transact-SQL の文字列関数、文字列変数、リテラルは 16K まで使用できます。

set string_rtruncation がオンの場合、insert または update で文字列がトランケートされると、ユーザはエラーを受け取ります。しかし、表示される文字列がトランケートされても、Adaptive Server はエラーをレポートしません。次に例を示します。

```
select replicate("a", 16383) + replicate("B", 4000)
```

この例は、全長が 20383 になっても、結果文字列は 16K に制限されていることを示します。

text 関数および image 関数

text 関数は、text、image、unitext データを操作します。select 文によって取り出される text、image、unitext データの量を制限するには、set textsize オプションを使用してください。

注意　このほかに、text、image、および unitext データを扱うのに、*@@textcolid*、*@@textdbid*、*@@textobjid*、*@@textptr*、*@@textsize* グローバル変数を使用できます。

たとえば、textptr 関数を使用して、title_id BU78322 を持つ、text カラムの copy を blurbs テーブルで検索します。16 バイトのバイナリ文字列であるテキスト・ポインタは、ローカル変数 *@val* に挿入され、readtext コマンドのパラメータとして指定されます。オフセットを 1 を指定された readtext は、2 番目のバイトで始まる 5 バイトを返します。カラムに含まれるデータの一部だけを取り出す場合は、readtext で text、unitext、image の値を取り出すことができます。

```
declare @val binary(16)
select @val = textptr(copy) from blurbs
where au_id = "486-29-1786"
readtext blurbs.copy @val 1 5
```

textptr は 16 バイトの varbinary 文字列を返します。上記の例のように、この文字列をローカル変数に入れて、参照によって使用することをおすすめします。

textptr を使用する上記の declare の例に対して、*@@textptr* グローバル変数を使用する代替方法を次に示します。

```
readtext texttest.blurb @@textptr 1 5
```

注意 文字列関数 patindex および datalength を text カラム、image カラム、unitext カラムに使用可能です。

unitext カラムでの *readtext* の使用

カラムに含まれるデータの特定の部分だけの場合は、readtext コマンドで text、unitext、および image の値を取り出すことができます。readtext コマンドには、テーブルおよびカラムの名前、テキスト・ポインタ、カラム内の開始オフセット、検索する文字数またはバイト数の情報が必要です。ビューの text、unitext、または image カラムでは、readtext を使用できません。

readtext コマンドの unitext カラムでの使用の詳細については、『リファレンス・マニュアル：コマンド』を参照してください。

集合関数

集合関数は、クエリ結果に新しいカラムとして表示される合計値を生成します。それらは select リスト、または select 文の having 句か、サブクエリで使用できます。where 句内では使用できません。

集合関数は、char データ型値に適用される場合、値を暗黙的に varchar に変換して、後続ブランクをすべて削除します。同様に、unichar データ型の値は暗黙的に univarchar データ型へ変換されます。

制限事項

- クエリ内の各集合には、それぞれのワーク・テーブルが必要です。したがって、集合関数のクエリでは、クエリに許可されたワーク・テーブルの最大数 (46) を超える数の集合を使用できません。
- カーソルの select 句に集合関数を含めた場合、このカーソルは更新できません。
- 集合関数は sysprocesses や syslocks などの仮想テーブルには使用できません。

group by 句を指定した集合関数

集合関数は、多くの場合、テーブルをグループに分割する group by 句を指定します。集合関数は、グループごとに 1 つの値を作成します。group by を使用しないと、select リスト内の集合関数は、テーブル内のすべてのローを処理するか、または where 句よって定義されたローのサブセットを処理するかに関係なく、結果的に 1 つの値を作成します。

集合関数と null 値

集合関数は、特定カラムにある null 以外の値の合計値を計算します。ansinull オプションを off (デフォルト値) に設定すると、集合関数で null が検出されても警告は出されません。ansinull オプションをオンに設定すると、集合関数で null が検出されたときに、クエリから次の SQLSTATE 警告が返されます。

```
Warning- null value eliminated in set function
```

ベクトルおよびスカラ集合

集合関数は、テーブル内のすべてのローに適用できます。この場合、単一の値であるスカラ集合が作成されます。集合関数は、指定したカラムまたは式で同じ値を持つすべてのローにも (group by と、オプションで having 句を使用して) 適用できます。この場合は、グループごとに 1 つの値、ベクトル集合が作成されます。集合関数の結果は新しいカラムとして表示されます。

スカラ集合関数内にベクトル集合関数をネストできます。次に例を示します。

```
select type, avg(price), avg(avg(price))
from titles
group by type
type
------------  ------------  ------------
UNDECIDED              NULL         15.23
business              13.73         15.23
mod_cook              11.49         15.23
popular_comp          21.48         15.23
psychology            13.50         15.23
trad_cook             15.96         15.23

(6 rows affected)
```

group by 句は、ベクトル集合に適用されます。この例では、avg(price) です。スカラ集合 avg(avg(price)) は、titles テーブルの種類別平均価格の平均値です。

標準的な SQL では、select リストに集合関数が含まれる場合、すべての select リストのカラムに集合関数が適用されるか、またはこれらのカラムが group by リストに含まれている必要があります。Transact-SQL には、このような制限はありません。

例 1 は、標準の制限が適用される select 文を示します。例 2 では、同じ文の select リストに別の項目 (title_id) が追加されています。表示内容の違いを明確に示すため、order by も追加されています。これらの「余分な」カラムは、having 句でも参照できます。

例 1

```
select type, avg(price), avg(advance)
from titles
group by type

type
------------  ------------  ------------
UNDECIDED              NULL         NULL
```

```
business              13.73     6,281.25
mod_cook              11.49     7,500.00
popular_comp          21.48     7,500.00
psychology            13.50     4,255.00
trad_cook             15.96     6,333.33

(6 rows affected)
```

例 2

group by の後で、カラム名または他の式 (カラム見出しまたはエイリアスを除く) のどちらかを使用できます。

group by カラムの null 値は、1 つのグループにまとめられます。

```
select type, title_id, avg(price), avg(advance)
from titles
group by type
order by type
```

```
type         title_id
-----------  --------    ---------- ---------
UNDECIDED       MC3026      NULL       NULL
business        BU1032      13.73   6,281.25
business        BU1111      13.73   6,281.25
business        BU2075      13.73   6,281.25
business        BU7832      13.73   6,281.25
mod_cook        MC2222      11.49   7,500.00
mod_cook        MC3021      11.49   7,500.00
popular_comp    PC1035      21.48   7,500.00
popular_comp    PC8888      21.48   7,500.00
popular_comp    PC9999      21.48   7,500.00
psychology      PS1372      13.50   4,255.00
psychology      PS2091      13.50   4,255.00
psychology      PS2106      13.50   4,255.00
psychology      PS3333      13.50   4,255.00
psychology      PS7777      13.50   4,255.00
trad_cook       TC3218      15.96   6,333.33
trad_cook       TC4203      15.96   6,333.33
trad_cook       TC7777      15.96   6,333.33
```

例 3

select 文の compute clause は、ロー集合を使用して合計値を作成します。ロー集合を使用すると、1 つのコマンドでディテール・ローと合計ローを検索できます。例 3 に、この機能を示します。

```
select type, title_id, price, advance
from titles
where type = "psychology"
order by type
compute sum(price), sum(advance) by type
```

```
type         title_id    price       advance
-----------  -------     ----------  ---------
psychology   PS1372          21.59   7,000.00
psychology   PS2091          10.95   2,275.00
```

```
psychology  PS2106          7.00   6,000.00
psychology  PS3333          19.99   2,000.00
psychology  PS7777          7.99   4,000.00
                            sum     sum
                            ------- ----------
                            67.52   21,275.00
```

例 3 の表示内容と、compute を使用しない例 (例 1 と 例 2) の表示内容の違いに注意してください。

ロー集合としての集合関数

ロー集合関数は、クエリ結果に追加ローとして表示される合計値を生成します。

集合関数をロー集合として使用するには、次を使用します。

```
Start of select statement
compute row_aggregate(column_name)
        [, row_aggregate(column_name)]...
    [by column_name [, column_name]...]
```

それぞれの意味は、次のとおりです。

- *column_name* – カラム名。カッコで囲む必要があります。sum と avg で使用できるのは、真数値、概数値、通貨カラムだけです。

  1 つの compute clause は、同じ関数をいくつかのカラムに適用できます。複数の関数を使用するには、複数の compute clause を使用してください。

- by は、ローの集約値がサブグループについて計算されることを示します。by 項目の値が変更されると、常にローの集約値が生成されます。by を使用する場合は、order by を使用する必要があります。

  by の後に複数の項目をリストすると、グループがサブグループに分割され、グループ化される各レベルに関数が適用されます。

ロー集合を使用すると、1 つのコマンドでディテール・ローと合計ローを検索できます。集合関数は通常、テーブル内の指定されたすべてのローに対して、または各グループに対して 1 つの値を生成し、この合計値は新しいカラムとして表示されます。

次の例は、この違いを示しています。

```
select type, sum(price), sum(advance)
from titles
where type like "%cook"
group by type

type
----------  ----------  ----------------
mod_cook         22.98         15,000.00
trad_cook        47.89         19,000.00

(2 rows affected)
```

```
select type, price, advance
from titles
where type like "%cook"
order by type
compute sum(price), sum(advance) by type

type         price          advance
----------   ----------   ----------------
mod_cook          2.99          15,000.00
mod_cook         19.99                0.00
             sum          sum
             ----------   ----------------
                  22.98          15,000.00
type           price       advance
----------   ----------   ----------------
trad_cook        11.95           4,000.00
trad_cook        14.99           8,000.00
trad_cook        20.95           7,000.00
             sum          sum
             ----------   ----------------
                  47.89          19,000.00
(7 rows affected)
type         price        advance
----------   ----------   ----------------
mod_cook          2.99          15,000.00
mod_cook         19.99                0.00

Compute Result:
---------------------- -----------------
                  22.98          15,000.00
type           price          advance
----------   ----------   ----------------
trad_cook        11.95           4,000.00
trad_cook        14.99           8,000.00
trad_cook        20.95           7,000.00

Compute Result:
---------------------- -----------------
                  47.89          19,000.00
(7 rows affected)
```

compute 句内のカラムは select リストになければなりません。

select リストでのカラムの指定順序は、compute clause の集合の順序を上書きします。次に例を示します。

```
create table t1 (a int, b int, c int null)
insert t1 values(1,5,8)
insert t1 values(2,6,9)

(1 row affected)

compute sum(c),  max(b), min(a)
```

```
select a, b, c from t1

a           b           c
----------- ----------- -----------
          1           5           8
          2           6           9

Compute Result:
----------- ----------- -----------
          1           6          17
```

ansinull オプションを off (デフォルト値) に設定すると、ロー集合で null が検出されても警告は出されません。ansinull オプションをオンに設定すると、ロー集合で null が検出されたときに、クエリから次の SQLSTATE 警告が返されます。

```
Warning- null value eliminated in set function
```

compute clause の出力を結果テーブルに保存する方法がないため、compute clause と同じ文中で select into を使用することはできません。

統計集合関数

統計集合関数で数値データの統計的分析を行うことができます。「集合関数」(488 ページ) を参照してください。

これらの関数は、クエリの group by 句の指定に従ってローのグループの値を計算できる集合関数です。max や min などのその他の基本的な集合関数と同様に、これらの計算は入力データ内の null 値を無視します。また、分析される式のドメインに関係なく、分散と標準偏差の計算では必ず IEEE (Institute of Electrical and Electronics Engineers) の倍精度浮動小数点数が使用されます。

分散関数または標準偏差関数への入力が空のデータ・セットである場合、これらの関数は null 値を返します。分散関数または標準偏差関数への入力が単一の値である場合、これらの関数は 0 を返します。

統計集合は、avg 集合に類似しています。

- 構文は次のとおりです。

  statistical_agg_function_name ([all | distinct] *expression*)

- numeric データ型の式のみが有効です。
- null 値は計算には関与しません。
- データが計算に関与しない場合、結果は null のみです。
- distinct 句または all 句は、式の前に置かれます (デフォルトは allです)。
- 統計集合をベクトル集合 (group by を含む)、スカラ集合 (group by を含まない) として、または compute 句で使用できます。

ただし avg 集合関数とは異なり、結果は次のようになります。

- 常に float データ型 (つまり、倍精度浮動小数点)、ただし avg 集合の結果のデータ型は式のデータ型と同じです (例外あり)。
- 単一データ・ポイントでは 0.0 です。

標準偏差を計算する式

Adaptive Server が分散と標準偏差を定義するのに使用する式を確認するには、『リファレンス・マニュアル：ブロック』の stddev_samp、stddev_pop var_samp、および var_pop のリファレンス・ページを参照してください。

計算式は類似していますが、次の異なる目的のために使用されます。

- var_samp および stddev_samp － 母集団の標本、つまりサブセットを全母集団の代表として評価する際に使用されます。
- var_pop および stddev_pop － 母集団で使用できるデータがすべてある場合、または *n* が非常に大きくて、*n* と *n-1* の差が小さい場合に使用されます。

算術関数

算術関数は、算術データの演算に共通して必要な値を返します。

また、各関数には、指定のデータ型に暗黙的に変換できる引数を指定することもできます。たとえば、概数値型を受け入れる関数は、整数値型も受け入れます。Adaptive Server により、引数は指定のデータ型に変換されます。「内部変換エラー」(507 ページ) を参照してください。

ドメイン・エラーまたは範囲エラーを処理するために、エラー・トラップが用意されています。ドメイン・エラーの処理方法を指定するために、arithabort と arithignore を使用できます。

表 16-1 は、floor、ceiling、round 算術関数を使用した例を示します。

表 16-1: 算術関数の例

文	結果
select floor(123)	123
select floor(123.45)	123.000000
select floor(1.2345E2)	123.000000
select floor(-123.45)	-124.000000
select floor(-1.2345E2)	-124.000000
select floor($123.45)	123.00
select ceiling(123.45)	124.000000
select ceiling(-123.45)	-123.000000
select ceiling(1.2345E2)	124.000000
select ceiling(-1.2345E2)	-123.000000
select ceiling($123.45)	124.00
select round(123.4545, 2)	123.4500
select round(123.45, -2)	100.00
select round(1.2345E2, 2)	123.450000
select round(1.2345E2, -2)	100.000000

日付関数

日付関数は、算術計算を実行し、datetime、bigtime、bigdatetime、smalldatetime、date、time 値に関するデータを表示します。クエリの select 句または where 句で使用できる。

1753 年 1 月 1 日より後の値には、datetime データ型を使用します。0001 年 1 月 1 日から 9999 年 1 月 1 日までの日付には、date を使用します。一重または二重の引用符で囲んでください。Adaptive Server では、さまざまな種類の日付フォーマットを識別できます。データ型の詳細については、『リファレンス・マニュアル：ビルディング・ブロック』を参照してください。

これがデフォルトの表示フォーマットです。

```
Apr 15 2010 10:23PM
```

それぞれの日付は、Adaptive Server に指定できる略語を使用する部分に分割されます。表 16-2 に、各日付要素、省略形がある場合はその省略形と、日付要素に使用できる整数値を示します。

表 16-2: 日付要素

日付要素	省略形	値
year	yy	1753 ～ 9999 (datetime) 1900 ～ 2079 (smalldatetime) 0001 － 9999 (date)
quarter	qq	1 ～ 4
month	mm	1 ～ 12
week	wk	1 ～ 54
day	dd	1 ～ 31
dayofyear	dy	1 ～ 366
weekday	dw	1 ～ 7 (日曜日～土曜日)
hour	hh	0 ～ 23
minute	mi	0 ～ 59
second	ss	0 ～ 59
millisecond	ms	0 ～ 999
microsecond	us	0 ～ 999999

たとえば、datediff 関数で、指定した 1 番目と 2 番目の 2 つの日付の差を、日付要素に指定した単位で算出します。結果は、*date2* から *date1* を引いた値に等しい符号付きの整数値が日付要素に返されます。

このクエリは、pubdate と 2010 年 11 月 30 日の間の日数を算出します。

```
select newdate = datediff(day, pubdate, getdate())
    "Nov 30 2010")
from titles
pubdate                  newdate
------------------------ ---------------
Jun 12 2006 12:00AM                1632
Jun  9 2005 12:00AM                2000
Jun 30 2005 12:00AM                1979
Jun 22 2004 12:00AM                2352
Jun  9 2006 12:00AM                1635
Jun 15 2004 12:00AM                2356
...
```

dateadd 関数で、指定した日付にある日付間隔 (整数として指定) を加算します。たとえば、titles テーブルのすべての本の発行が 3 日遅くなった場合、次の文を使用して新しい発行日を得ることができます。

```
select dateadd(day, 3, pubdate)
from titles


---------------------
Jun 15 2006 12:00AM
Jun 12 2005 12:00AM
Jul  3 2005 12:00AM
```

```
Jun 25 2004 12:00AM
Jun 12 2006 12:00AM
Jun 21 2004 12:00AM
...
```

データ型変換関数

データ型変換は、式のデータ型を別のデータ型への変更と、date および time 情報の表示フォーマットを再フォーマットします。

Adaptive Server では、「暗黙の変換」と呼ばれる特定のデータ型の変換が実行されます。たとえば、char 式と datetime 式、または smallint 式と int 式、あるいは長さの異なる char 式を比較するときに、Adaptive Server がデータ型を別のデータ型へ自動的に変換します。

「明示的な変換」と呼ばれるその他の変換については、データ型変換をするためにデータ型変換関数を使用する必要があります。たとえば、数値型を連結するには、その前に数値型を文字型に変換しておかなければなりません。date を datetime に明示的に変換しようとしたときに、その値が datetime の範囲外 (たとえば、“Jan 1, 1000”) の場合は、変換が許可されず、エラー・メッセージが表示されます。「明示的変換のための convert 関数」(500 ページ) を参照してください。

暗黙的にも明示的にも、変換できないデータ型があります。たとえば、smallint または binary データは datetime に変換できません。表 16-3 と表 16-4 は、各データ型変換が暗黙的または明示的に実行されるか、あるいはそのテーブルでサポートされていないかを示しています。

- E − 明示的なデータ型変換が必要です。
- I − 明示的にも暗黙的にも変換できます。
- U − データ型は変換できません。

表 16-3: 暗黙的、明示的、およびサポートされていないデータ型変換

変換前	binary	varbinary	bit	[n]char	[n]varchar	datetime	smalldatetime	bigdatetime	bigtime	tinyint	smallint	unsigned smallint	int	unsigned int
binary	–	I	I	I	I	U	U	I	I	I	I	I	I	I
varbinary	I	–	I	I	I	U	U	I	I	I	I	I	I	I
bit	I	I	–	I	I	U	U	U	U	I	I	I	I	I
[n]char	I	I	E	–	I	I	I	I	I	E	E	E	E	E
[n]varchar	I	I	E	I	–	I	I	I	I	E	E	E	E	E
datetime	I	I	U	I	I	–	I	I	I	U	U	U	U	U
smalldatetime	I	I	U	I	I	I	–	I	I	U	U	U	U	U
bigdatetime	I	I	U	I	I	I	I	-	I	U	U	U	U	U
bigtime	I	I	U	I	I	I	I	I	-	U	U	U	U	U
tinyint	I	I	I	E	E	U	U	U	U	–	I	I	I	I
smallint	I	I	I	E	E	U	U	U	U	I	–	I	I	I
unsigned smallint	I	I	I	E	E	U	U	U	U	I	I	–	I	I
int	I	I	I	E	E	U	U	U	U	I	I	I	–	I
unsigned int	I	I	I	E	E	U	U	U	U	I	I	I	I	–
bigint	I	I	I	E	E	U	U	U	U	I	I	I	I	I
unsigned bigint	I	I	I	E	E	U	U		U	I	I	I	I	I
decimal	I	I	I	E	E	U	U	U	U	I	I	I	I	I
numeric	I	I	I	E	E	U	U	U	U	I	I	I	I	I
float	I	I	I	E	E	U	U	U	U	I	I	I	I	I
real	I	I	I	E	E	U	U	U	U	I	I	I	I	I
money	I	I	I	I	I	U	U	U	U	I	I	I	I	I
smallmoney	I	I	I	I	I	U	U	U	U	I	I	I	I	I
text	U	U	U	E	E	U	U	U	U	U	U	U	U	U
unitext	E	E	E	E	E	U	U	U	U	U	U	U	U	U
image	E	E	U	U	U	U	U	U	U	U	U	U	U	U
unichar	I	I	E	I	I	I	I	I	I	E	E	E	E	E
univarchar	I	I	E	I	I	I	I	I	I	E	E	E	E	E
date	I	I	U	I	I	I	U	I	U	U	U	U	U	U
time	I	I	U	I	I	I	U	I	I	U	U	U	U	U

表 16-4: 暗黙的、明示的、およびサポートされていないデータ型変換

変換前	bigint	unsigned bigint	decimal	numeric	float	real	money	smallmoney	text	unitext	image	unichar	univarchar	date	time
binary	I	I	I	I	I	I	I	I	U	I	I	I	I	I	I
varbinary	I	I	I	I	I	I	I	I	U	I	I	I	I	I	I
bit	I	I	I	I	I	I	I	I	U	U	U	E	E	U	U
[n]char	E	E	E	E	E	E	E	E	I	I	I	I	I	I	I
[n]varchar	E	E	E	E	E	E	E	E	I	I	I	I	I	I	I
datetime	U	U	U	U	U	U	U	U	U	U	U	I	I	I	I
smalldatetime	U	U	U	U	U	U	U	U	U	U	U	I	I	I	I
bigdatetime	U	U	U	U	U	U	U	U	U	U	U	I	I	I	I
bigtime	U	U	U	U	U	U	U	U	U	U	U	I	I	U	I
tinyint	I	I	I	I	I	I	I	I	U	U	U	E	E	U	U
smallint	I	I	I	I	I	I	I	I	U	U	U	U	E	U	U
unsigned smallint	I	I	I	I	I	I	I	I	U	U	U	E	E	U	U
int	I	I	I	I	I	I	I	I	U	U	U	E	E	U	U
unsigned int	I	I	I	I	I	I	I	I	U	U	U	E	E	U	U
bigint	–	I	I	I	I	I	I	I	U	U	U	E	E	U	U
unsigned bigint	I	–	I	I	I	I	I	I	U	U	U	E	E	U	U
decimal	I	I	–	I	I	I	I	I	U	U	U	E	E	U	U
numeric	I	I	I	–	I	I	I	I	U	U	U	E	E	U	U
float	I	I	I	I	–	I	I	I	U	U	U	E	E	U	U
real	I	I	I	I	I	–	I	I	U	U	U	E	E	U	U
money	I	I	I	I	I	I	–	I	U	U	U	E	E	U	U
smallmoney	I	I	I	I	I	I	I	–	U	U	U	E	E	U	U
text	U	U	U	U	U	U	U	U	–	I	U	E	E	U	U
unitext	U	U	U	U	U	U	U	U	I	–	I	U	U	U	U
image	U	U	U	U	U	U	U	U	U	I	–	E	E	U	U
unichar	E	E	E	E	E	E	E	E	I	I	I	–	I	I	I
univarchar	E	E	E	E	E	E	E	E	I	I	I	I	–	I	I
date	U	U	U	U	U	U	U	U	U	U	U	I	I	–	I
time	U	U	U	U	U	U	U	U	U	U	U	I	I	I	–

明示的変換のための convert 関数

汎用変換関数の convert はさまざまなデータ型の変換や、日時のデータの新しい表示のフォーマットの指定に使用します。構文は次のとおりです。

convert(*datatype* [(*length*) | (*precision*[, *scale*])] [null | not null],
expression [, *style*])

次は、select リストでの convert の使用例です。

```
select title, convert(char(5), total_sales)
from titles
where type = "trad_cook"
title
-----------------------------------     -----
Onions, Leeks, and Garlic: Cooking
    Secrets of the Mediterranean          375
Fifty Years in Buckingham Palace
    Kitchens                            15096
Sushi, Anyone?                           4095

(3 rows affected)
```

長さ、または精度と位取りのいずれかを指定しなければならないデータ型があります。長さを指定しない場合には、Adaptive Server によって文字データとバイナリ・データにデフォルトの長さ 30 が使用されます。精度または位取りを指定しない場合には、Adaptive Server によってデフォルトの 18 が精度に、0 が位取りに使用されます。

データ型変換のガイドラインと制約

文字型データの非文字型への変換

文字データは、完全に新しい型に有効な文字からなる場合、通貨、日付／時刻、真数値、または概数値型などの非文字型に変換できます。先行ブランクは無視されます。ただし、1 つまたは複数のブランクで構成される char データ型の式を datetime データ型の式に変換した場合は、Adaptive Server によって、ブランクがデフォルトの datetime 値「Jan 1, 1900」に変換されます。

受け入れられない文字がデータに含まれている場合には構文エラーが生成されます。次に、構文エラーの原因となる文字の例をいくつか示します。

- 整数データ内のカンマまたは小数点
- 通貨データ内のカンマ
- 真数値データまたは概数値データ、あるいはビット・ストリーム・データの文字
- 日付**データ**および**時刻**データ内の月名のスペルミス

unichar/univarchar と datetime/smalldatetime 間の暗黙的な変換がサポートされています。

ある文字型から他の文字型への変換

マルチバイト文字セットをシングルバイト文字セットに変換する場合、該当するシングルバイトのない文字は疑問符に変換されます。

text カラムおよび unitext カラムは、char、nchar、varchar、unichar、univarchar、または nvarchar に明示的に変換できます。変換は、サーバの論理ページ・サイズの最大カラム・サイズによって決定される character 型の最大長に制限されます。長さを指定しない場合、変換された値はデフォルトの長さである 30 バイトになります。

数値型データの文字型への変換

真数値型データおよび概数値型データは、文字型に変換できます。新しい型が短すぎて文字列が収まらない場合は、空き領域不足エラーが発生します。たとえば、次に示す変換では 1 文字の型に 5 文字の文字列を格納しようとしています。

```
select convert(char(1), 12.34)

Insufficient result space for explicit conversion
of NUMERIC value '12.34' to a CHAR field.
```

float データを文字型に変換する場合、新しい型の長さは少なくとも 25 文字にする必要があります。

注意　変換を行うときには、convert または cast よりもstr 関数の使用をおすすめします。この関数には変換を詳細に制御する機能があり、エラーを回避できます。

---

unitext へ、または unitext からの変換

unitext は、別の文字またはバイナリのデータ型から暗黙的に変換できます。unitext から別のデータ型、または別のデータ型から unitext に明示的に変換することもできます。ただし、変換結果は変換先のデータ型の最大長に制約されます。unitext 値が Unicode 文字境界上の変換先バッファに収まらない場合、データはトランケートされます。enable surrogate processing を設定している場合は、unitext 値はサロゲート・ペア値の中央でトランケートされません。つまり、データ変換後に返されるバイト数が少なくなることがあります。たとえば、テーブル tb の unitext カラム ut に文字列 “U+0041U+0042U+00c2” (U+0041 は Unicode 文字 “A”) が保管されている場合、次のクエリはサーバの文字セットが UTF-8 のときに値 “AB” を返します。これは、U+00C2 が 2 バイトの UTF-8 0xc382 に変換されるからです。

```
select convert(char(3), ut) from tb
```

現在、alter table modify コマンドには、変更対象カラムとして text、image、または unitext カラムを指定できません。text カラムを unitext カラムにマイグレートするには、bcp out を使用して既存のデータをコピーし、unitext カラムのテーブルを作成してから、bcp in を使用して新しいテーブルにデータを挿入します。この方法でマイグレートするには、bcp の呼び出し時に -Jutf8 オプションを指定する必要があります。

通貨型との変換中の丸め処理

money と smallmoney 型は小数点以下 4 桁を保管しますが、表示の都合上、最も近い小数点以下第 2 位 (.01) に丸められます。通貨型に変換されたデータは、4 桁まで丸められます。

通貨型から変換されたデータでは、可能であれば同じ丸め処理が行われます。新しい型が小数点以下の桁数が 3 桁未満の真数値である場合、データは新しい型の位取りに従って丸められます。たとえば、$4.50 が整数に変換されると 5 になります。

```
select convert(int, $4.50)
 -----------
           5
```

money または smallmoney に変換されたデータの単位は、セントなどの補助貨幣単位ではなく、ドルなどの主要貨幣単位とみなされます。たとえば英語の場合、整数値 5 は、5 セントではなく 5 ドルに等しい通貨に変換されます。

日付と時刻情報の変換

日付として認識されるデータは、datetime、smalldatetime、date、または time に変換できます。誤った月の名前は、構文エラーの原因になります。また、データ型の許容範囲外の日付は、算術オーバフロー・エラーの原因になります。

datetime 値が smalldatetime 値に変換される場合、この値は最も近い分数まで丸められます。

「日付フォーマットの変更」(505 ページ) を参照してください。

数値型間での変換

データは、ある数値型から別の数値型に変換できます。新しい型が、データを収容できるだけの精度または位取りがない真数値型である場合は、エラーが発生します。

たとえば、整数を予期する組み込み関数への引数として float 値または数値を指定すると、その float 値または数値の値はトランケートされます。ただし Adaptive Server では、小数部分を持つ数値は暗黙的に変換されず、位取りエラー・メッセージが返されます。たとえば、Adaptive Server では、小数部分を持つ数値の場合はエラー 241、他のデータ型が渡された場合はエラー 257 が返されます。

数値変換によって生じたエラーの Adaptive Server による処理方法を決定するには、arithabort と arithignore の各オプションを使用してください。

バイナリ型と整数型の変換

binary 型と varbinary 型は、“0x” プレフィクスとその後に続く数字と文字の文字列からなる 16 進数に似たデータを格納します。

これらの文字列はプラットフォームによって解釈が異なります。たとえば、文字列「0x0000100」は、バイト 0 を最上位とみなす (リトル・エンディアン) マシンでは 65536 を表し、バイト 0 を最下位とみなす (ビッグ・エンディアン) マシンでは 256 を表します。

バイナリ型は、convert 関数を使用すると明示的に、使用しないと暗黙的に整数型に変換されます。データが小さすぎて新しい型を収められない場合は、その「0x」プレフィクスと埋め込みゼロが削除されます。データが長すぎる場合は、トランケートされます。

convert と暗黙的データ型変換では、プラットフォームに適合した形式でバイナリ・データが評価されます。このため、結果はプラットフォームによって異なることがあります。hextoint 関数は、プラットフォームに依存しない 16 進文字列から整数への変換に使用します。また、inttohex 関数は、プラットフォームに依存しない整数から 16 進数の値への変換に使用します。hextobigint 関数は、プラットフォームに依存しない 16 進文字列から 64 ビット整数への変換に使用します。また、biginttohex 関数は、プラットフォームに依存しない 64 ビット整数から 16 進数の値への変換に使用します。

バイナリ型と数値型または 10 進数型の間のデータ変換

binary と varbinary データ文字列では、0x に続く最初の 2 つの値は binary 型を表します。“00” は正の数値を表し、“01” は負の数値を表します。binary または varbinary 型のデータを numeric または decimal に変換する場合は、必ず “0x” の後ろに “00” または “01” の値を指定してください。このように指定されていないと変換は失敗します。

次の例では、binary 型データを numeric 型データに変換する方法を示します。

```
select convert(numeric
(38, 18),0x00000000000000006b14bd1e6eea0000000000000000000000000000000000)

----------
123.456000
```

同じデータを numeric 型から binary 型に変換するには、次を使用します。

```
select convert(binary,convert(numeric(38, 18), 123.456))

------------------------------------------------------------------
0x00000000000000006b14bd1e6eea0000000000000000000000000000000000
```

image カラムからバイナリ型への変換

convert 関数を使用して、image カラムを binary または varbinary に変換できます。binary データ型の最大長は制限されています。この最大長は、サーバの論理ページ・サイズの最大カラム・サイズによって決定します。長さを指定しない場合、変換された値はデフォルトの長さである 30 文字になります。

他の型から *bit* への変換

真数値型および概数値型は、bit 型に暗黙的に変換できます。文字型からの変換には、明示的な convert 関数が必要です。

変換される式は、数字、小数点、通貨記号、プラス符号またはマイナス符号だけで構成します。他の文字があると、構文エラーが発生します。

0 に対応する bit は 0 です。他のすべての数字に対応する bit は 1 です。

16 進数のデータの変換

複数のプラットフォームにわたって信頼性のある変換結果を得るには、hextoint と inttohex 関数を使用します。

類似した関数の hextobigint と biginttohex は、64 ビットの整数との変換に使用できます。

hextoint は、“0x” プレフィクスの有無にかかわらず、数字と A ～ F の大文字および小文字から構成されているリテラルまたは変数を受け付けます。次の hextoint の使用方法はすべて有効です。

```
select hextoint("0x00000100FFFFF")
select hextoint("0x00000100")
select hextoint("100")
```

hextoint により、16 進データから “0x” プレフィクスが削除されます。データが 8 桁を超える場合、hextoint はそれをトランケートします。8 桁に満たない場合、hextoint は右揃えを行い、ゼロを埋め込みます。次に hextoint は、プラットフォームに依存しない、16 進数のデータと同じ整数を返します。上記の式はすべて、hextoint 関数を実行するプラットフォームに関係なく、同じ値の 256 を返します。

inttohex 関数は整数データを受け付け、“0x” プレフィクスがない 8 文字の「**16** 進文字列」を返します。inttohex は、どのプラットフォームを使用しているかに関係なく、常に同じ結果を返します。

bigtime データと bigdatetime データの変換

暗黙的または明示的な変換は、精度の低下によってデータ・ロスが発生する場合に許可されます。

プライマリ・フィールドが一致しないデータ型間で暗黙的な変換が行われると、データのトランケーション、デフォルト値の挿入、またはエラー・メッセージが発生する可能性があります。たとえば、bigdatetime 値を date 値に変換すると、時刻部分がトランケートされ、日付要素のみが残ります。bigtime 値を bigdatetime 値に変換すると、新しい bigdatetime 値の日付要素に、デフォルト日付の Jan 1, 0001 が追加されます。date 値を bigdatetime 値に変換すると、bigdatetime 値の時刻部分に、デフォルト時刻の 00:00:00.000000 が追加されます。

null 値の変換

null から not null への変換と not null から null への変換を実行するには、convert 関数を使用します。

日付フォーマットの変更

convert 関数の *style* パラメータにより、datetime または smalldatetime データを char または varchar に変換するときに、日付の表示フォーマットを指定できます。*style* パラメータとして指定する数字の引数によって、データの表示方法が決まります。年は 2 または 4 桁で入力できます。100 を *style* 値に加算すると、西暦を表す 4 桁の年 (*yyyy*) を表示できます。

表 16-5 は、*style* に指定する値と使用できる日付フォーマットを示します。*style* を smalldatetime 型で使用する場合、秒またはミリ秒を含む形式では、そこに 0 が表示されます。

表のキー　“mon” は月名、“mm” は数字の月または分です。“HH” は 24 時間形式の値、“hh” は 12 時間形式の値です。最後のロー 23 には、フォーマット内で日付と時刻を区切るリテラル “T” が含まれます。

表 16-5: style パラメータによる日付フォーマットの変換

世紀なし (yy)	世紀あり (yyyy)	標準	出力
-	0 または 100	デフォルト	*mon dd yyyy hh:mm* AM (または PM)
1	101	USA	*mm/dd/yy*
2	2	SQL 規格	*yy.mm.dd*
3	103	イギリス／フランス	*dd/mm/yy*
4	104	ドイツ	*dd.mm.yy*
5	105		*dd-mm-yy*
6	106		*dd mon yy*
7	107		*mon dd, yy*
8	108		*HH:mm:ss*
-	9 または 109	デフォルト + ミリ秒	*mon dd yyyy hh:mm:sss* AM (または PM)
10	110	USA	*mm-dd-yy*
11	111	日本	*yy/mm/dd*
12	112	ISO	*yymmdd*
13	113		*yy/dd/mm*

世紀なし (yy)	世紀あり (yyyy)	標準	出力
14	114		*mm/yy/dd*
15	115		*dd/yy/mm*
-	16 または 116		*mon dd yyyy HH:mm:ss*
17	117		*hh:mmAM*
18	118		*HH:mm*
19			*hh:mm:ss:zzzAM*
20			*HH:mm:ss:zzz*
21			*yy/mm/dd HH:mm:ss*
22			*yy/mm/dd hh:mm* AM (または PM)
	23		*yyyy-mm-ddTHH:mm:ss*

デフォルト値を使用する、0 または 100、および 9 または 109 の形式は、必ず西暦 (*yyyy*) を返します。

この例では、現在の日付がスタイル 3 の *dd/mm/yy* の形式に変換されます。

```
select convert(char(12), getdate(), 3)
```

date データを文字型に変換するときは、表 16-5 のスタイル番号の 1 ～ 7 (101 ～ 107) または 10 ～ 12 (110 ～ 112) を使って表示フォーマットを指定します。デフォルト値は 100 (*mon dd yyyy hh:mi*AM (または PM)) です。時刻部分を含むスタイルに date データを変換する場合は、時刻部分はデフォルト値の 0 になります。time データを文字型に変換するときは、スタイル番号の 8 または 9 (108 または 109) を使って表示フォーマットを指定します。デフォルトは 100 (*mon dd yyyy hh:mi*AM (または PM)) です。日付部分を含むスタイルに time データを変換する場合は、デフォルト日付の Jan 1, 1900 が表示されます。

注意 *style* 引数に null を指定した convert では、*style* 引数を指定しない convert と同じ結果が返されます。次に例を示します。

```
select convert(datetime, "01/01/01")
------------

Jan 1 2001 12:00AM

select convert(datetime, "01/01/01", NULL)
------------

Jan 1 2001 12:00AM
```

内部変換エラー

算術オーバフロー・エラーと 0 による除算エラー

0 による除算エラーは、Adaptive Server が数値をゼロで除算しようとすると発生します。算術オーバフロー・エラーは、新しい型に、結果が収まるだけの小数の桁がない場合に発生します。これは、次の変換時に発生します。

- 精度または位取りが少ない真数値型への明示的または暗黙的な変換
- 通貨型または日付／時刻型に受け入れられる範囲を外れるデータの明示的または暗黙的変換
- hextoint を使用した 4 バイト以上の記憶領域を必要とする 16 進文字列の変換

算術オーバフロー・エラーと 0 による除算エラーは、暗黙的または明示的変換のどちらで発生したかに関係なく、重大エラーとみなされます。Adaptive Server によるこれらのエラーの処理方法を指定するには、arithabort arith_overflow オプションを使用してください。デフォルト設定の arithabort arith_overflow on では、エラーが発生したトランザクション全体がロールバックされます。arithabort arith_overflow on が設定されていると、トランザクションを含まないバッチでエラーが発生した場合、エラーの発生前にバッチに含まれていたコマンドはロールバックしません。ただし Adaptive Server は、バッチでエラーを起こした文のあとでは、どの文も実行しません。arithabort arith_overflow off に設定すると、Adaptive Server は、エラーの原因となる文をアボートして、トランザクションまたはバッチの他の文の処理を継続します。*@@error* グローバル変数を使用して、文の結果をチェックできます。

Adaptive Server がこれらのエラー発生後にメッセージを表示するかどうかを決定するには、arithignore arith_overflow オプションを使用してください。デフォルト設定の off では、0 による除算エラーまたは精度のロスが発生すると、警告メッセージが表示されます。arithignore arith_overflow on を設定すると、これらのエラーが発生しても警告メッセージは表示されません。オプションの arith_overflow キーワードを省略しても影響はありません。

位取りのエラー

明示的な変換によって位取りのロスが発生すると、この結果は警告なしにトランケートされます。たとえば、float、numeric、または decimal 型を integer 型に明示的に変換すると、Adaptive Server では、結果を整数にして、小数点の右側にあるすべての数字をトランケートするものと想定されます。

numeric 型または decimal 型への暗黙の変換時に、位取りのロスがあると、位取りエラーが発生します。arithabort numeric_truncation オプションを使用して、エラーの重大度を調べます。デフォルト設定の arithabort numeric_truncation on は、エラーを起こした文をアボートしますが、トランザクションまたはバッチ内のその他の文の処理は継続します。arithabort numeric_truncation off を設定した場合、Adaptive Server はクエリ結果をトランケートして処理を継続します。

注意 エントリ・レベルの ANSI SQL に準拠する場合は、次のオプションを設定してください。

- arithabort arith_overflow off
- arithabort numeric_truncation on
- arithignore off

ドメイン・エラー

convert 関数は、定義されている範囲以外の引数が指定されると、ドメイン・エラーとなります。このエラーは、ほとんど発生しません。

セキュリティ関数

セキュリティ関数を使用すると、セキュリティ・サービスおよびユーザ定義の役割についての情報を表示できます。

ユーザ・パーミッションの管理の詳細については、『セキュリティ管理ガイド』を参照してください。

XML 関数

XML 関数により、XML を Adaptive Server データベースで管理できます。XML 関数については、『XML サービス』で説明しています。

ユーザ定義関数

独自のスカラ Transact-SQL 関数を作成して保存するには、create function コマンドを使用します。次を含めることができます。

- 関数に対してローカルであるデータ変数およびカーソルを定義する declare 文
- 関数にローカルなオブジェクトに割り当てられた値 (たとえば、select コマンドまたは set コマンドを含むテーブルに対してローカルなスカラおよび変数に割り当てられた値)
- 関数で宣言、オープン、クローズ、割り付け解除されたローカル・カーソルを参照するカーソル操作
- フロー制御文
- set オプション (関数のスコープでのみ有効)

次を含めることはできません。

- データをクライアントに返す select または fetch 文
- insert、update、または delete 文
- dbcc、dump、load などのユーティリティ・コマンド
- print 文
- rand、rand2、getdate、または newid を参照する文

ローカル変数にのみ値を割り当てる select または fetch 文を含めることができます。

『リファレンス・マニュアル：コマンド』を参照してください。

注意 Java メソッドが指定する値を返す Transact-SQL 関数を作成することもできます。Transact-SQL ラッパを Java メソッドに追加するには、create function (SQLJ) コマンドを使用します。『Java in the Adaptive Server Database』と『リファレンス・マニュアル：コマンド』を参照してください。

第 17 章

ストアド・プロシージャの使用

「ストアド・プロシージャ」とは、SQL 文やフロー制御言語の集まりに名前を付けたものです。頻繁に使用する機能のストアド・プロシージャを作成して、パフォーマンスの向上に役立てることができます。Adaptive Server には、管理作業やシステム・テーブルの更新を実行する「システム・プロシージャ」も備えています。

また、拡張ストアド・プロシージャを作成し、それを使用して Adaptive Server から手続き型言語関数を呼び出すこともできます。「第 18 章 拡張ストアド・プロシージャの使用」を参照してください。

ストアド・プロシージャの動作

ストアド・プロシージャによって、次のことができます。

- パラメータを取得する
- 他のプロシージャを呼び出す
- 呼び出し側のプロシージャまたはバッチにステータス値を返し、正常に終了したかどうかと、失敗した場合にはその理由を表示する
- 呼び出し側のプロシージャまたはバッチにパラメータ値を返す
- リモートの Adaptive Server 上で実行する

ストアド・プロシージャを作成すると、SQL の機能、効率、柔軟性が非常に強化されます。コンパイルされたプロシージャによって、SQL 文およびバッチのパフォーマンスが大幅に向上します。さらに、使用中のサーバとリモート・サーバの両方がリモート・ログインできるように設定されている場合は、他の Adaptive Server のストアド・プロシージャも実行できます。削除、更新、挿入などのイベントがローカルに発生した場合には、リモート・サーバでプロシージャを実行するトリガをローカルの Adaptive Server 上で作成できます。

ストアド・プロシージャは、プリコンパイルされているという点で、通常の SQL 文および SQL 文のバッチとは異なります。プロシージャを初めて実行したとき、Adaptive Server のクエリ・プロセッサによってそのプロシージャが分析され、正常に完了した後に「システム・テーブル」に保管される実行プランが作成されます。それ以降は、プロシージャは保管されたプランに従って実行されます。クエリ処理作業の多くはクエリ・プロセッサによって既に実行されているため、ストアド・プロシージャはほとんど即時に実行されます。

ユーザの利便性のため、Adaptive Server では、数多くのストアド・プロシージャを提供しています。これらのストアド・プロシージャは “sp_” で始まる名前で sybsystemprocs データベースに格納されており、システム・テーブルにデータを挿入、更新、削除、レポートを行うことから、「システム・プロシージャ」と呼ばれます。

Sybase が提供するすべてのシステム・プロシージャの完全リストについては、『リファレンス・マニュアル：プロシージャ』を参照してください。

例

パラメータなどの特別な機能を除いた、簡単なストアド・プロシージャを作成する構文は、次のとおりです。

```
create procedure procedure_name
    as SQL_statements
```

ストアド・プロシージャはデータベース・オブジェクトなので、識別子の規則に従って命名します。

create 文を除けば、どのような種類の SQL 文をいくつでも、ストアド・プロシージャに含めることができます。「ストアド・プロシージャに関連する規則」(539 ページ) を参照してください。プロシージャは、データベース内のすべてのユーザ名を 1 つの文でリストできるのと同じ程度に単純に作成できます。

```
create procedure namelist
as select name from sysusers
```

ストアド・プロシージャを実行するには、execute キーワードとプロシージャ名を使用します。または、プロシージャ名が単独で Adaptive Server に送信される場合や、プロシージャ名がバッチ内の最初の文である場合は、プロシージャ名だけを使用します。たとえば、次のいずれの方法でも namelist を実行できます。

```
namelist
execute namelist
exec namelist
```

リモートの Adaptive Server でストアド・プロシージャを実行する場合、サーバ名を含める必要があります。リモート・プロシージャ・コールの構文は次のとおりです。

execute *server_name*.[*database_name*].[*owner*].*procedure_name*

「プロシージャのリモート実行」(530 ページ) を参照してください。

ストアド・プロシージャが「デフォルト・データベース」にある場合は、データベース名を省略できます。また、データベース所有者 (dbo) がプロシージャを所有している場合や自分がプロシージャを所有している場合には、所有者名を省略できます。プロシージャを実行するには「パーミッション」が必要です。

プロシージャには複数の文を含めることができます。

```
create procedure showall as
select count(*) from sysusers
select count(*) from sysobjects
select count(*) from syscolumns
```

このプロシージャが実行されると、プロシージャ内に文が出現する順序で各コマンドの結果が表示されます。

```
showall
------------

           5

(1 row affected)

------------
          88

(1 row affected)

------------
         349

(1 row affected, return status = 0)
```

create procedure コマンドが正常に実行されると、プロシージャ名は sysobjects に、そのプロシージャの「ソース・テキスト」は syscomments に格納されます。

ストアド・プロシージャを作成すると、そのプロシージャを説明する「ソース・テキスト」が、syscomments システム・テーブルの text カラムに格納されます。syscomments からこの情報を削除しないでください。削除すると、Adaptive Server の今後のアップグレードで問題が発生する場合があります。sp_hidetext を使用して、syscomments 内でテキストを暗号化します。『リファレンス・マニュアル：プロシージャ』および「コンパイル済みオブジェクト」(3 ページ) を参照してください。

プロシージャのソース・テキストを表示するには、次のように sp_helptext を使用できます。

```
sp_helptext showall
# Lines of Text
--------------
              1

(1 row affected)

text
----------------------------------------
create procedure showall as
select count(*) from sysusers
select count(*) from sysobjects
select count(*) from syscolumns

(1 row affected, return status = 0)
```

遅延名前解決 (これによりまだ存在しないオブジェクトを参照するストアド・プロシージャを作成できます) のプロシージャを作成すると、syscomments 内のテキストは select * 拡張を実行することなく保存されます。プロシージャの実行が最初に成功した後、Adaptive Server によって select * 拡張が実行され、プロシージャのテキストが拡張されたテキストで更新されます。select * 拡張の実行後にテキストが更新されるため、次の例に示すように、最終的なテキストには拡張された select * が含まれます。

```
create table t (a int, b int)
---------

set deferred_name_resolution on
---------------

create proc p as select * from t
------------------

sp_helptext p
-------------

 # Lines of Text
---------------
              1

(1 row affected)
text
------------------------------------------------------------------
------------------------------------------------------------------
------------------------------------------------------------------
----------------------------------------------------------
create proc p as select * from t
(1 row affected)
(return status = 0)

exec p
```

```
------------
a           b
----------- -----------

(0 rows affected)
(return status = 0)

sp_helptext p
-------------

 # Lines of Text
---------------
              1

(1 row affected)
text
------------------------------------------------------------------
------------------------------------------------------------------
------------------------------------------------------------------
---

 /* Adaptive Server has expanded all '*' elements in the
following statement */

create proc p as select t.a, t.b from t
(1 row affected)
(return status = 0)
```

パーミッション

ストアド・プロシージャは、セキュリティのメカニズムとして機能させることもできます。ユーザは、ストアド・プロシージャ内で参照されるテーブルまたはビューのパーミッションや特定のコマンドを実行するパーミッションを持っていなくても、ストアド・プロシージャを実行するパーミッションを付与してもらうことができるからです。『システム管理ガイド 第 1 巻』の「第 17 章 ユーザ・パーミッションの管理」を参照してください。

ストアド・プロシージャのソース・テキストに対する不正なアクセスを防ぐには、syscomments テーブルの text カラム上の select パーミッションを、所有者とシステム管理者に限定します。この制限を実施するには、Adaptive Server を「評価済み設定」で実行する必要があります。この制限を有効にするには、システム・セキュリティ担当者が sp_configure の allow select on syscomments.text column パラメータをリセットする必要があります。『システム管理ガイド 第 1 巻』の「第 5 章 設定パラメータ」を参照してください。

ストアド・プロシージャのソース・テキストへのアクセスを保護する別の方法は、sp_hidetext を使用してソース・テキストを隠すことです。『リファレンス・マニュアル：プロシージャ』を参照してください。

パフォーマンス

データベースが変更されると、クエリ・プランを再コンパイルして、データのテーブルのアクセスに使用していた最初のクエリ・プランを最適化できます。これにより、すべてのストアド・プロシージャとトリガを検索し、削除して、再作成しなくて済みます。次の例では、titles テーブルをアクセスするすべてのストアド・プロシージャとトリガが、次に実行されるときに再コンパイルされるようにマーク付けされます。

```
sp_recompile titles
```

『リファレンス・マニュアル：プロシージャ』を参照してください。

ストアド・プロシージャの作成と実行

現在のデータベース内でプロシージャを作成できます。

15.5 より前のバージョンの Adaptive Server では、プロシージャの作成時にすべての参照先のオブジェクトが存在する必要がありました。遅延名前解決機能により、オブジェクト (ユーザが作成したデータ型オブジェクトを除く) をストアド・プロシージャの初回実行時に解決できるようになりました。

遅延名前解決は、サーバ・レベルで動作する deferred name resolution 設定パラメータ、または接続レベルで動作する set deferred_name_resolution パラメータを使用します。

デフォルトの動作では、実行前にオブジェクトが解決されます。設定オプション deferred name resolution または set パラメータを使用して明示的に遅延名前解決を指定する必要があります。

『システム管理ガイド 第 1 巻』および『リファレンス・マニュアル：コマンド』を参照してください。

create procedure コマンドを発行するパーミッションはデフォルトではデータベース所有者に付与されていて、他のユーザに譲渡できます。

遅延名前解決の使用

この機能が set オプションの deferred_name_resolution を使用してアクティブになっている場合、または設定パラメータの deferred name resolution を使用してグローバルにアクティブになっている場合、プロシージャ内のオブジェクトは作成時ではなく実行時に解決されます。このオプションを使用すると、プロシージャの作成時に存在しないオブジェクトを参照するプロシージャを作成できます。

たとえば、deferred_name_resolution を使用して、まだ存在しないテーブルを参照するプロシージャを作成できます。次の例では、deferred_name_resolution を指定しないでプロシージャを作成しようとしています。

```
select * from non_existing_table
-----------
error message


Msg 208, Level 16, State 1:
Line 1:
non_existing_table not found.Specify owner.objectname or use
sp_help to check whether the object exists (sp_help may produce
lots of output).
```

このオプションを使用すると、オブジェクトが存在しないことが原因でエラーが発生することなく、プロシージャを作成できます。

```
set deferred_resolution_on
-----------------
create proc p as select * from non_existing_table
-----------------
```

注意　deferred_name_resolution を使用した場合、ユーザ定義データ型は実行時に解決されません。ユーザ定義データ型は作成時に解決されます。したがって、解決に失敗した場合、プロシージャを作成できません。

作成時にオブジェクトが解決されるということは、オブジェクト解決エラーが作成時だけではなく実行時にも発生することを意味します。

パラメータ

「パラメータ」は、ストアド・プロシージャの引数です。オプションで create procedure 文に、1 つまたは複数のパラメータを宣言できます。create procedure 文の中で指定した各パラメータの値は、プロシージャの実行時に、ユーザが指定します。

パラメータ名は、先頭に @ 記号を付け、識別子の規則に従った名前を付ける必要があります。「識別子」(10 ページ) を参照してください。パラメータ名は、それを作成するプロシージャに対してローカルなので、同じパラメータ名を、別のプロシージャでも使用できます。句読点 (データベース名または所有者名で修飾されたオブジェクト名など) を含むパラメータ値は、一重引用符か二重引用符で囲みます。パラメータ名の最大長は、@ 記号も含めて 255 バイトです。

パラメータには、text、unitext、または image 以外のシステム・データ型、あるいはユーザ定義データ型を指定する必要があります。また、データ型によっては、カッコ内にデータの長さ、または精度と桁数を指定する必要があります。

次に pubs2 データベースのストアド・プロシージャを示します。作家の姓と名前がパラメータとして指定され、プロシージャによってその作家が書いたすべての本の名前とそれぞれの本の出版社が表示されます。

```
create proc au_info @lastname varchar(40),
  @firstname varchar(20) as
select au_lname, au_fname, title, pub_name
from authors, titles, publishers, titleauthor
where au_fname = @firstname
and au_lname = @lastname
and authors.au_id = titleauthor.au_id
and titles.title_id = titleauthor.title_id
and titles.pub_id = publishers.pub_id
```

ここで次のように au_info を実行します。

```
au_info Ringer, Anne

au_lname au_fname title                 pub_name
-------- -------- --------------------- ----------
Ringer   Anne     The Gourmet Microwave Binnet & Hardley
Ringer   Anne     Is Anger the Enemy?   New Age Books
(2 rows affected, return status = 0)
```

次のストアド・プロシージャは、システム・テーブルに問い合わせます。テーブル名がパラメータとして指定され、プロシージャによって、テーブル名、インデックス名、インデックス ID が表示されます。

```
create proc showind @table varchar(30) as
select table_name = sysobjects.name,
index_name = sysindexes.name, index_id = indid
from sysindexes, sysobjects
where sysobjects.name = @table
and sysobjects.id = sysindexes.id
```

table_name などのカラム見出しは、結果を読みやすくするために追加されています。このストアド・プロシージャを実行するために使用可能な構文形式を次に示します。

```
execute showind titles
exec showind titles
execute showind @table = titles
execute GATEWAY.pubs2.dbo.showind titles
showind titles
```

exec または execute のない上記の最後の構文形式は、文がバッチ内の唯一の文であるかバッチ内の最初の文である場合にだけ使用できます。

titles がパラメータとして指定された場合の pubs2 データベースでの showind の実行結果を次に示します。

```
table_name  index_name  index_id
----------  ----------  ----------
titles      titleidind      0
titles      titleind      2

(2 rows affected, return status = 0)
```

パラメータを “*@parameter = value*” の形式で指定する場合、どの指定順でもかまいません。別の形式で指定する場合、パラメータを create procedure 文における順序で指定する必要があります。1 つの値を “*@parameter = value*” の形式で指定する場合は、そのあとに続くパラメータもすべてこの形式で指定してください。

このプロシージャでは、salesdetail テーブルの qty カラムのデータ型を表示しています。

```
create procedure showtype @tabname varchar(18), @colname
varchar(18) as
  select syscolumns.name, syscolumns.length,
  systypes.name
  from syscolumns, systypes, sysobjects
  where sysobjects.id = syscolumns.id
  and @tabname = sysobjects.name
  and @colname = syscolumns.name
  and syscolumns.type = systypes.type
```

このプロシージャを実行するときに、次のように名前で指定する場合は *@tabname* と *@colname* を、create procedure 文の記述順と異なる順で指定できます。

```
exec showtype
@colname = qty , @tabname = salesdetail
```

値式を使用するストアド・プロシージャでは、case 式を使用できます。次の例は、titles テーブル内のすべての本の販売部数をチェックします。

```
create proc booksales @titleid tid
as
select title, total_sales,
case
when total_sales != null then "Books sold"
when total_sales = null then "Book sales not available"
end
from titles
where @titleid = title_id
```

次に例を示します。

```
booksales MC2222
title                              total_sales
-------------------------          -----------
Silicon Valley Gastronomic Treats    2032 Books sold

(1 row affected)
```

デフォルト・パラメータ

create procedure 文では、パラメータのデフォルト値を割り当てることができます。デフォルト値には任意の定数を指定でき、ユーザが引数を指定しない場合にプロシージャに使用されます。

パラメータとして指定した出版社によって出版された本を書いた作家名をすべて表示するプロシージャを次に示します。このプロシージャは Algodata Infosystems 社から本を出版した作家を表示します。

```
create proc pub_info
  @pubname varchar(40) = "Algodata Infosystems" as
select au_lname, au_fname, pub_name
from authors a, publishers p, titles t, titleauthor ta
where @pubname = p.pub_name
and a.au_id = ta.au_id
and t.title_id = ta.title_id
and t.pub_id = p.pub_id
```

デフォルト値が埋め込みブランクまたは句読表示を含む文字列である場合は、その値を一重または二重の引用符で囲む必要があります。

pub_info を実行するときは、パラメータ値として任意の出版社名を指定できます。パラメータを指定しない場合は、Adaptive Server によって、デフォルト値である Algodata Infosystems が使用されます。

```
exec pub_info
au_lname        au_fname      pub_name
--------------  ------------  --------------------
Green           Marjorie      Algodata Infosystems
Bennet          Abraham       Algodata Infosystems
O'Leary         Michael       Algodata Infosystems
MacFeather      Stearns       Algodata Infosystems
Straight        Dick          Algodata Infosystems
Carson          Cheryl        Algodata Infosystems
Dull            Ann           Algodata Infosystems
Hunter          Sheryl        Algodata Infosystems
Locksley        Chastity      Algodata Infosystems

(9 rows affected, return status = 0)
```

次のプロシージャ showind2 は、*@table* パラメータのデフォルト値として "titles" を割り当てます。

```
create proc showind2
@table varchar(30) = titles as
select table_name = sysobjects.name,
    index_name = sysindexes.name, index_id = indid
from sysindexes, sysobjects
where sysobjects.name = @table
and sysobjects.id = sysindexes.id
```

table_name などのカラム見出しによって、結果の表示が読みやすくなります。showind2 によって、authors テーブルが以下のように表示されます。

```
showind2 authors

table_name  index_name      index_id
----------- -------------   ---------
authors     auidind                1
authors     aunmind                2

(2 rows affected, return status = 0)
```

ユーザが値を指定しない場合は、Adaptive Server によってデフォルト値である *titles* が使用されます。

```
showind2

table_name  index_name  index_id
----------- ----------- ---------
titles      titleidind         1
titles      titleind           2

(2 rows affected, return status = 0)
```

期待されたパラメータが指定されず、create procedure 文にデフォルト値が指定されていない場合、Adaptive Server は、プロシージャが期待するパラメータをリストするエラー・メッセージを表示します。

ストアド・プロシージャでのデフォルト・パラメータの使用

パラメータにデフォルトを使用するストアド・プロシージャを作成すると、ユーザがそのストアド・プロシージャを発行するときにパラメータ名のスペルを間違えた場合、Adaptive Server はデフォルト値を使用してストアド・プロシージャを実行します。エラー・メッセージは発行しません。たとえば、次のようなプロシージャを作成したとします。

```
create procedure test @x int = 1
as select @x
```

次の結果が返されます。

```
exec test @x = 2
go
-----------------
                2
```

しかし、次のように、このストアド・プロシージャに間違ったパラメータを渡した場合は、間違った結果セットが返されますが、エラー・メッセージは発行されません。

```
exec test @z = 4
go
-----------
          1
(1 row affected)
(return status = 0)
```

デフォルト・パラメータとしての null

create procedure 文では、null を個々のパラメータのデフォルト値として宣言できます。次に例を示します。

```
create procedure procedure_name
    @param datatype [ = null ]
    [, @param datatype [ = null ]]...
```

ユーザがパラメータを指定しないと、Adaptive Server はエラー・メッセージを表示しないでストアド・プロシージャを実行します。

ユーザがパラメータを指定しない場合、プロシージャ定義は、パラメータ値が null であるかを確認して、実行するアクションを指定できます。次に例を示します。

```
create procedure showind3
@table varchar(30) = null as
if @table is null
    print "Please give a table name."
else
   select table_name = sysobjects.name,
     index_name = sysindexes.name,
     index_id = indid
   from sysindexes, sysobjects
   where sysobjects.name = @table
   and sysobjects.id = sysindexes.id
```

ユーザがパラメータを指定しなかった場合は、Adaptive Server によって、プロシージャからのメッセージが画面に表示されます。

デフォルト値を null に設定する他の例については、sp_helptext を使用して、システム・プロシージャのソース・テキストを参照してください。

デフォルト・パラメータでのワイルドカード文字の使用

プロシージャ内で like キーワードを持つパラメータを使用する場合は、デフォルト値にワイルドカード文字 (%、_、[]、[^]) を使用できます。

たとえば、前述の showind を次のように修正すると、パラメータが指定されない場合にシステム・テーブルについての情報を表示するようにできます。

```
create procedure showind4
@table varchar(30) = "sys%" as
select table_name = sysobjects.name,
    index_name = sysindexes.name,
    index_id = indid
from sysindexes, sysobjects
where sysobjects.name like @table
and sysobjects.id = sysindexes.id
```

複数のパラメータの使用

次に、両方のパラメータにワイルドカード文字が含まれるデフォルトを使用しているストアド・プロシージャ au_info の変形を示します。

```
create proc au_info2
  @lastname varchar(30) = "D%",
  @firstname varchar(18) = "%" as
select au_lname, au_fname, title, pub_name
from authors, titles, publishers, titleauthor
where au_fname like @firstname
and au_lname like @lastname
and authors.au_id = titleauthor.au_id
and titles.title_id = titleauthor.title_id
and titles.pub_id = publishers.pub_id
```

パラメータを指定しないで au_info2 を実行すると、姓が “D” で始まるすべての作家が表示されます。

```
au_info2

au_lname au_fname title                        pub_name
-------- -------  ------------------------    -------------
Dull     Ann      Secrets of Silicon Valley   Algodata Infosystems
DeFrance Michel   The Gourmet Microwave        Binnet & Hardley

(2 rows affected)
```

複数のパラメータにデフォルト値を使用できる場合は、最後のパラメータから順に省略できます。指定されたデフォルト値が null 以外の場合は、途中のパラメータを省略することはできません。

注意　パラメータを *@parameter = value* の形式で指定する場合は、順不同で指定できます。デフォルト値が指定されていれば、パラメータを省略することもできます。1 つの値を *@parameter = value* の形式で指定する場合は、そのあとに続くパラメータもすべてこの形式で指定してください。

---

2 つのパラメータのデフォルト値が定義されている場合に 2 番目のパラメータを省略している例を示します。姓が “Ringer” であるすべての作家の本および出版社が次のように表示されます。

```
au_info2 Ringer

au_lname   au_fname    title                    Pub_name
--------   --------    ---------------------    ------------
Ringer     Anne        The Gourmet Microwave    Binnet & Hardley
Ringer     Anne        Is Anger the Enemy?      New Age Books
Ringer     Albert      Is Anger the Enemy?      New Age Books
Ringer     Albert      Life Without Fear        New Age Books
```

ストアド・プロシージャを実行し、プロシージャが期待していたパラメータ数より多いパラメータを指定すると、Adaptive Server は、超過したパラメータを無視します。たとえば、pubs2 データベースに対して sp_helplog を実行すると、次のように表示されます。

```
sp_helplog

In database 'pubs2', the log starts on device 'pubs2dat'.
```

誤って無意味なパラメータを指定しても、sp_helplog の結果は同じです。

```
sp_helplog one, two, three

In database 'pubs2', the log starts on device 'pubs2dat'.
```

SQL は自由な形式の言語です。1 行内のワード数や、改行の仕方に規則はありません。ストアド・プロシージャ、コマンドの順で発行すると、Adaptive Server は、プロシージャ、コマンドの順で実行しようとします。たとえば、次のコマンドを発行します。

```
sp_help checkpoint
```

Adaptive Server は、sp_help の出力を返し、checkpoint コマンドを実行します。プロシージャ・パラメータに区切り識別子を出力すると、期待どおりの結果が得られない場合があります。

ストアド・プロシージャにおけるラージ・オブジェクトの *text*、*unitext*、*image* データ型の使用

Adaptive Server バージョン 15.7 以降では、ローカル変数に対してラージ・オブジェクト (LOB) の text、image、または unitext データ型を宣言し、その変数を入力パラメータとしてストアド・プロシージャに渡すことができます。15.7 よりも前のバージョンでは、ストアド・プロシージャの text、image、または unitext データの名前付きパラメータが現在のページ・サイズ (2、4、8、または 16KB) を超過することができませんでした。

この例は、ストアド・プロシージャに LOB データ型を使用します。

1 table_1 を作成するとします。

```
create table t1 (a1 int, a2 text)
    insert into t1 values(1, "aaaa")
    insert into t1 values(2, "bbbb")
    insert into t1 values(3, "cccc")
```

2 LOB ローカル変数をパラメータとして使用してストアド・プロシージャを作成します。

```
create procedure my_procedure @loc text
    as select @loc
```

3　ローカル変数を宣言して、ストアド・プロシージャを実行します。

```
declare @a text
select @a = a2 from t1 where a1 = 3
exec my_procedure @a
--------------------------
cccc
```

いくつかの制限が適用されます。LOB データ型の制限は、次のとおりです。

- ストアド・プロシージャの出力パラメータとして使用できない。
- convert() 関数を使用したデータ型変換に使用できない。
- 複写でサポートされない。

プロシージャ・グループ

create procedure と execute 文のプロシージャ名の後にオプションでセミコロンと整数を使用すると、同じ名前のプロシージャをグループ化でき、それらを削除する場合は 1 つの drop procedure 文でまとめて実行できます。

同じアプリケーションで使用するプロシージャは、この方法でグループ化されます。たとえば、orders;1、orders;2 のような一連のプロシージャを作成したとします。グループ全体を削除するには、以下を使用します。

```
drop proc orders
```

名前にセミコロンと数字を付けてグループ化されたプロシージャは、単独では削除できません。たとえば、次の文は使用できません。

```
drop proc orders;2
```

評価済み設定で Adaptive Server を実行する場合は、プロシージャのグループ化を行わないでください。これによって、すべてのストアド・プロシージャはユニークなオブジェクト識別子を持つことができ、個別にプロシージャの削除ができます。プロシージャのグループ化を禁止するには、システム・セキュリティ担当者が allow procedure grouping 設定パラメータをリセットする必要があります。『システム管理ガイド 第 1 巻』の「第 5 章 設定パラメータ」を参照してください。

create procedure での with recompile の使用

create procedure 文では、SQL 文のすぐ前にオプションの with recompile 句を置きます。この句は、Adaptive Server に、このプロシージャのプランを保持しないように指示します。新しいプランは、プロシージャが実行されるたびに作成されます。

with recompile 句がない場合、Adaptive Server は作成した実行プランを保管します。通常、この実行プランで十分です。しかし、データの変更、または後続の実行のために指定されるパラメータ値の変更によって、最初にプロシージャが実行されたときに Adaptive Server が作成した実行プランと異なる実行プランが必要になることがあります。このような場合、Adaptive Server には新しい実行プランが必要です。

新しいプランを作成するには、create procedure 文で with recompile 句を使用します。『リファレンス・マニュアル：コマンド』を参照してください。

with *recompile* の使用

Adaptive Server バージョン 12.5 のマニュアルでは、ストアド・プロシージャの実行のたびに操作するデータが異なる場合、with recompile を使用してストアド・プロシージャを作成することを推奨しています。これにより、ストアド・プロシージャは、前回の実行時のプランを使用するのではなく、実行のたびに再コンパイルされます。Adaptive Server バージョン 15.0 ではこのオプションの重要性が高くなり、バージョン 15.0.2 以降の遅延コンパイルの導入によりきわめて重要になりました。

ストアド・プロシージャの複数のコピーが同時にプロシージャ・キャッシュに格納される場合、ストアド・プロシージャの同時実行により、前回の実行時のクエリ・プランを使用する問題がさらに深刻になる可能性があります。ストアド・プロシージャの実行のたびにまったく異なるデータ・セットが使用された場合、プロシージャ・キャッシュには、ストアド・プロシージャのコピー (それぞれがまったく異なるプランを使用) が 2 つ以上格納されます。その後のストアド・プロシージャの実行では、MRU (最も最近に使用された) アルゴリズムに基づいて選択されたコピーが使用されます。

この問題により、同じストアド・プロシージャを実行するたびにパフォーマンスが大幅に変動する可能性があります。同じ状況は Adaptive Server 12.5 で発生する場合がありますが、プロシージャはマジック・ナンバーを使用して最適化されるため、プランは同じである可能性があります。したがって、パフォーマンスが大幅に変動する可能性は低くなります。

トラブルシューティング

ストアド・プロシージャのパフォーマンス問題のトラブルシューティングを行うときは、with recompile を使用して、テスト中に使用される各ストアド・プロシージャが再コンパイルされるようにします。これにより、テスト中に前回のコンパイル時のプランが使用されることはなくなります。

execute での with recompile の使用

execute 文では、オプションの with recompile 句はどのパラメータの後に置いてもかまいません。この句は、Adaptive Server に新しいプランをコンパイルするように指示します。

データが大きく変更された場合、または指定したパラメータが変則的である場合、つまり、プロシージャとともに保管されているプランがプロシージャの実行に最適ではないと考えられる場合には、プロシージャを実行するときに with recompile 句を使用します。

execute procedure *with recompile* を何度も使用すると、プロシージャ・キャッシュのパフォーマンスが低下することがあります。*with recompile* を使用するたびに新しいプランが生成されるため、新しいプラン用に十分な領域がキャッシュにない場合は、役に立つパフォーマンス・プランが古いものとしてキャッシュから出される可能性があります。

create procedure 文で select * を使用した場合は、execute 文に with recompile 句があっても、プロシージャはテーブルに追加された新しいカラムを選択しません。プロシージャを削除して、再作成してください。

プロシージャ内でのプロシージャのネスト

あるストアド・プロシージャまたはトリガが別のストアド・プロシージャまたはトリガを呼び出すと、ネストが発生します。ネスト・レベルは、呼び出されるプロシージャまたはトリガの実行が始まると増加し、呼び出されるプロシージャ、またはトリガの実行が完了すると減少します。ネスト・レベルは、キャッシュされる文が作成されたときも 1 つ増えます。ネストが最大レベルである 16 を超えると、プロシージャは失敗します。現在のネスト・レベルは、*@@nestlevel* グローバル変数に格納されます。

プロシージャ名か、または実際のプロシージャ名の代わりに変数名によって、別のプロシージャを呼び出すことができます。次に例を示します。

```
create procedure test1 @proc_name varchar(30)
    as exec @proc_name
```

ストアド・プロシージャ内でのテンポラリ・テーブルの使用

ストアド・プロシージャ内でテンポラリ・テーブルを作成したり使用したりできますが、テンポラリ・テーブルは、それを作成するストアド・プロシージャが実行されている間しか存在しません。プロシージャが終了すると、テンポラリ・テーブルは Adaptive Server によって自動的に削除されます。1 つのプロシージャで次のことができます。

- テンポラリ・テーブルの作成
- データの挿入、更新、または削除
- テンポラリ・テーブルでのクエリの実行
- テンポラリ・テーブルを参照する他のプロシージャの呼び出し

テンポラリ・テーブルを参照するプロシージャを作成するにはテンポラリ・テーブルが存在する必要があります。次の手順に従います。

1 create table 文または select into 文を使用して、テンポラリ・テーブルを作成します。次に例を示します。

```
create table #tempstores
    (stor_id char(4), amount money)
```

注意 set deferred_name_resolution を使用する場合、この手順は必要ありません。「遅延名前解決の使用」(516 ページ) を参照してください。

2 テンポラリ・テーブルにアクセスするプロシージャを作成します (テンポラリ・テーブルを作成するプロシージャではありません)。

```
create procedure inv_amounts as
    select stor_id, "Total Due" = sum(amount)
    from #tempstores
    group by stor_id
```

3 テンポラリ・テーブルを削除します。

```
drop table #tempstores
```

deferred_name_resolution を使用する場合、この手順は必要ありません。

4 テーブルの作成、および手順 2 で作成したプロシージャの呼び出しを実行するプロシージャを作成します。

```
create procedure inv_proc as
create table #tempstores
(stor_id char(4), amount money)
```

inv_proc プロシージャを実行するとテーブルが作成されますが、そのテーブルは、プロシージャの実行中にだけ存在します。次のように #tempstores テーブルに値を挿入するか、inv_amounts プロシージャを実行します。

```
insert #tempstores
select stor_id, sum(qty*(100-discount)/100*price)
from salesdetail, titles
where salesdetail.title_id = titles.title_id
group by stor_id, salesdetail.title_id
exec inv_amounts
```

#tempstores テーブルはもう存在していないので、上記のどちらの方法でもテーブルは作成できません。

ストアド・プロシージャの内部から create table tempdb..*tablename*... を使用して、# プレフィクスを付けずにテンポラリ・テーブルを作成できます。このようなテーブルはプロシージャが完了しても削除されないので、独立したプロシージャで参照できます。上記の手順に従ってこれらのテーブルを作成します。

ストアド・プロシージャ内のオプション設定

ストアド・プロシージャ内で、ほとんどすべての set コマンド・オプションを使用できます。set オプションは、プロシージャの実行中にだけ有効です。プロシージャがクローズされると以前の設定に戻ります。dateformat、datefirst、language、role の各オプションだけは、以前の設定には戻りません。

ただし、オブジェクト所有者だけが使用できる set オプション (identity_insert など) を、オブジェクト所有者以外のユーザが使おうとしても、そのストアド・プロシージャを実行できません。

クエリ最適化の設定

set export_options on を使用して、set plan optgoal や set plan optcriteria などの最適化設定をエクスポートできます。最適化設定は、ストアド・プロシージャに対してローカルではありません。ユーザ・セッション全体に適用されます。

注意　デフォルトでは、set export_options はログイン・トリガで有効です。

ストアド・プロシージャの引数

ストアド・プロシージャに指定できる引数は 2048 個までです。ただし、すべての引数の処理と、メモリとの間での引数値のコピーをクエリ処理エンジンが行う必要があるため、引数の多いプロシージャを実行した場合にはパフォーマンスが低下します。Sybase では、多数の引数を含むストアド・プロシージャを作成した場合、運用環境で実装する前にテストしておくことをおすすめします。

式、変数、引数の長さ

ストアド・プロシージャに渡せる式、変数、および引数の最大サイズは、どのページ・サイズでも 16384 バイト (16K) です。これは、文字またはバイナリ・データのいずれかです。writetext コマンドを使用せずに、この最大サイズまで変数およびリテラルを text カラムに挿入できます。

以前の一部のバージョンの Adaptive Server では、ストアド・プロシージャの式、変数、引数の最大サイズは 255 バイトでした。

最大長が短く制限されていた、以前のバージョンの Adaptive Server 用に作成されたスクリプトまたはストアド・プロシージャは、最大ページ・サイズが大きくなったため、長い文字列値を返すようになることがあります。

値が大きいため、Adaptive Server は文字列をトランケートする場合があります。また、他の変数に格納されたり、カラムまたは文字列に挿入された場合に文字列がオーバフローすることもあります。

既存のテーブルのカラムを修正して文字カラムを長くした場合、これらのカラム上のデータを操作するストアド・プロシージャを、新しい長さを反映するように変更する必要があります。

```
select datalength(replicate("x", 500)),
    datalength("abcdefgh....255 byte long string.."+
    "xxyyzz ... another 255 byte long string")

----------- -----------
255         255
```

ストアド・プロシージャの実行

一定時間後にストアド・プロシージャを実行することも、リモートからストアド・プロシージャを実行することもできます。

時間遅延後のプロシージャの実行

waitfor コマンドは、指定された時刻まで、または指定された時間が経過するまでプロシージャの実行を遅延させることができます。

たとえば、プロシージャ testproc を 30 分後に実行するには、次のように指定します。

```
begin
    waitfor delay "0:30:00"
    exec testproc
end
```

waitfor コマンドを実行した後は、設定した時刻またはイベントの発生まで、現在使用中の Adaptive Server への接続は使用できません。

プロシージャのリモート実行

ローカルの Adaptive Server から、リモートの Adaptive Server 上でプロシージャを実行できます。両方のサーバが適切に設定されていれば、識別子の一部としてサーバ名を使用するだけで、リモートの Adaptive Server 上で任意のプロシージャを実行できます。たとえば、GATEWAY という名前のサーバ上で remoteproc という名前のプロシージャを実行するには、次のようにします。

```
exec gateway.remotedb.dbo.remoteproc
```

次の例はいずれも、GATEWAY サーバ上の pubs2 データベースで namelist プロシージャを実行します。

```
execute gateway.pubs2..namelist
gateway.pubs2.dbo.namelist
exec gateway...namelist
```

上記の最後の例は、pub2 がデフォルト・データベースである場合にだけ使用できます。

『システム管理ガイド 第 1 巻』の「第 15 章 リモート サーバの管理」を参照してください。リモート・プロシージャ用の execute 文を含むバッチまたはプロシージャから、パラメータとして 1 つまたは複数の値をリモート・プロシージャに渡すことができます。リモートの Adaptive Server からの結果は、ローカルの端末に表示されます。

以降の項で説明されるプロシージャからのリターン・ステータスは、プロシージャの実行ステータスについての情報メッセージを取得および送信するために使用します。「リターン・ステータス」(532 ページ) を参照してください。

警告！ コンポーネント統合サービスが有効でない場合、Adaptive Server はリモート・プロシージャ・コール (RPC) をトランザクションの一部として扱いません。したがって、RPC のトランザクションの一部として実行し、その後でトランザクションをロールバックすると、Adaptive Server は、RPC が実行した変更をロールバックしません。コンポーネント統合サービスが有効な場合は、set transactional rpc と set cis rpc handling コマンドを使用して、トランザクション RPC を実行してください。『リファレンス・マニュアル：コマンド』を参照してください。

ストアド・プロシージャでの遅延コンパイル

Adaptive Server では、ストアド・プロシージャは最初の実行時に最適化されます (変数に渡された値が使用可能な限り)。

遅延コンパイルでは、Adaptive Server がローカル変数に値を割り当てる文やテンポラリ・テーブルを作成する文など、ストアド・プロシージャで前に登場した文を既に実行しています。これは、マジック・ナンバーではなく、既知の値とテンポラリ・テーブルに基づいて文が最適化されることを意味します。実際の値を使用することで、オプティマイザは特定のデータ・セットに対してストアド・プロシージャを実行するのに適したプランを選択できます。

Adaptive Server では、操作するデータがストアド・プロシージャのコンパイル時に使用されたデータと同じ場合は、同じプランをその後のストアド・プロシージャの実行に再利用できます。

15.0.2 より前のバージョンの Adaptive Server では、ストアド・プロシージャ内のすべての文がコンパイルされてから文が実行されていました。つまり、ローカル変数の実際の値またはストアド・プロシージャ内で作成されたテンポラリ・テーブルの情報は、最適化中には使用できませんでした。プランを含むコンパイルされたストアド・プロシージャは、プロシージャ・キャッシュに格納されました。

Adaptive Server 15.0.2 以降では、ローカル変数またはテンポラリ・テーブルを参照するストアド・プロシージャについて遅延コンパイルが使用されています。これを使用すると、ストアド・プロシージャは実行の準備ができるまでコンパイルされません。

プランは最初の実行に使用された値とデータ・セット用に最適化されるため、その後のストアド・プロシージャの実行で使用される値とデータ・セットが異なる場合、プランが最適ではない可能性があります。

ストアド・プロシージャから返される情報

ストアド・プロシージャは、次のタイプの情報を返します。

- リターン・ステータス ― ストアド・プロシージャが正常に終了したかどうかを示します。
- proc role 関数 ― プロシージャを実行したユーザが、sa_role、sso_role、または ss_oper 権限を所有しているかどうかをチェックします。
- リターン・パラメータ ― 呼び出し元にパラメータ値を返します。この後、呼び出し元は、条件文を使用して、返された値をチェックできます。

リターン・ステータスとリターン・パラメータによって、ストアド・プロシージャをモジュール化できます。いくつかのストアド・プロシージャによって使用される SQL 文のセットは、その実行ステータスやパラメータ値を呼び出し側のプロシージャに返す単一のプロシージャとして作成できます。たとえば、Adaptive Server が提供するシステム・プロシージャの多くには、特定のパラメータを有効な識別子として確認する別のプロシージャが含まれます。

リモートの Adaptive Server 上で実行されるストアド・プロシージャであるリモート・プロシージャ・コールは、ステータスとパラメータの両方を返します。execute 文の構文に、プロシージャ名だけでなく、サーバ名、データベース名、所有者名を指定すれば、次の各例はすべてリモートで実行されます。

リターン・ステータス

ストアド・プロシージャは、プロシージャが正常に終了したかどうか、または失敗した場合はその理由を示す「リターン・ステータス」を返します。この値はプロシージャが呼び出される時に変数に格納され、将来的に拡張される Transact-SQL 文で使用されます。失敗した場合のシステム定義のリターン・ステータスの値は -1 ～ -99 の範囲内で定義されています。この範囲以外であれば、リターン・ステータスの値を独自に定義できます。

リターン・ステータスを返す execute 文の形式を使用するバッチの例を次に示します。

```
declare @status int
execute @status = byroyalty 50
select @status
```

byroyalty プロシージャの実行ステータスは、*@status* 変数に保管されます。“50” は、titleauthor テーブルの royaltyper カラムに基づいて提供されたパラメータです。この例では、単に select 文による値が表示されますが、後の例では、この戻り値が条件付きの句で使用されます。

予約されたリターン・ステータス値

Adaptive Server では、正常終了を示すために 0 を、また、失敗のそれぞれの理由を示すために負の値である -1 ～ -99 を予約しています。表 17-1 で示すように、バージョン 12 以降では、0 および -1 ～ -14 の数字が現在使用されています。

表 17-1: 予約されたリターン・ステータス値

値	意味
0	プロシージャが正常に実行された
-1	オブジェクトがない
-2	データ型のエラー
-3	プロセスがデッドロックの被害対象として選択された
-4	パーミッション・エラー
-5	構文エラー
-6	その他のさまざまなユーザ・エラー
-7	領域不足などのリソース・エラー
-8	致命的ではない内部の問題
-9	システムの限界に達した
-10	致命的な内部不整合
-11	致命的な内部不整合
-12	テーブルまたはインデックスが破壊されている
-13	データベースが破壊されている
-14	ハードウェア・エラー

-15 ～ -99 の値は、今後の使用のために Adaptive Server に予約されています。

実行中に複数のエラーが発生した場合は、絶対値が最も高いステータスが返されます。

ユーザ生成の戻り値

ストアド・プロシージャ内で return 文にパラメータを追加して、ユーザ独自の戻り値を生成できます。0 ～ -99 の範囲外の任意の整数を使用できます。次の例では、本に有効な契約が結ばれている場合には 1 を返し、他のすべての場合には 2 を返します。

```
create proc checkcontract @titleid tid
as
if (select contract from titles where
        title_id = @titleid) = 1
   return 1
else
   return 2
```

次に例を示します。

```
checkcontract MC2222

(return status = 1)
```

次のストアド・プロシージャは、checkcontract を呼び出し、条件付きの句を使用してリターン・ステータスをチェックします。

```
create proc get_au_stat @titleid tid
as
declare @retvalue int
execute @retvalue = checkcontract @titleid
if (@retvalue = 1)
   print "Contract is valid."
else
    print "There is not a valid contract."
```

契約が有効な本の title_id を基準に get_au_stat を実行すると、次のようになります。

```
get_au_stat MC2222
Contract is valid
```

プロシージャ内での役割のチェック

ストアド・プロシージャにシステム管理や機密保護に関するタスクを実行させる場合は、特定の役割が与えられたユーザだけがそれらのタスクを実行するようにする必要があります。proc_role 関数を指定すると、プロシージャの実行時に役割をチェックでき、ユーザが指定された役割を持っている場合は、1 を返します。役割の名前は、sa_role、sso_role、および oper_role です。

ストアド・プロシージャ test_proc で proc_role を使用して、呼び出し側にシステム管理者であることを要求する場合の例を次に示します。

```
create proc test_proc
as
if (proc_role("sa_role") = 0)
begin
    print "You do not have the right role."
    return -1
end
else
    print "You have SA role."
    return(0)
```

次に例を示します。

```
test_proc

You have SA role.
```

リターン・パラメータ

ストアド・プロシージャが呼び出し元に情報を返す別の方法は、リターン・パラメータを使用することです。呼び出し元は、条件文を使用して、返された値をチェックできます。

create procedure 文と execute 文の両方で、output オプションがパラメータ名とともに指定されている場合は、プロシージャによって呼び出し側に値が返されます。呼び出し側は SQL バッチ、または別のストアド・プロシージャのいずれでもかまいません。返された値は、バッチや呼び出し側プロシージャ内に追加する文で使用できます。リターン・パラメータが、バッチの一部である execute 文で使用される場合は、バッチ内の次の文が実行される前に、その戻り値が見出し付きで表示されます。

次のストアド・プロシージャによって、2 つの整数の乗算が実行されます (3 番目の整数である *@result* は、output パラメータとして定義されています)。

```
create procedure mathtutor
@mult1 int, @mult2 int, @result int output
as
select @result = @mult1 * @mult2
```

乗算問題を解くために mathtutor を使用するには、*@result* 変数を宣言し、それを execute 文に含めます。output キーワードを execute 文に追加すると、リターン・パラメータの値が表示されます。

```
declare @result int
exec mathtutor 5, 6, @result output

(return status = 0)

Return parameters:

------------
          30
```

解答を得るために 3 つの整数を指定してこのプロシージャを実行しても、乗算の結果は出ません。プロシージャ内の select 文は値を代入しますが、出力は行いません。

```
mathtutor 5, 6, 32

(return status = 0)
```

output パラメータ値は、定数としてではなく、変数として渡してください。次の例では、*@guess* 変数を宣言して、*@result* で使用するために mathtutor に渡される値を保管します。Adaptive Server によって、次のようなリターン・パラメータが出力されます。

```
declare @guess int
select @guess = 32
exec mathtutor 5, 6,
@result = @guess output

(1 row affected)
(return status = 0)

Return parameters:

@result
-----------
        30
```

リターン・パラメータ値は、その値が変更されたかどうかに関係なく、常にレポートされます。次のことに注意してください。

- 上の例では、output パラメータ *@result* は "*@parameter* = *@variable*" の形式で渡す必要があります。出力パラメータが、最後に渡されたパラメータではない場合、後続のパラメータは "*@parameter* = *value*" 形式で渡します。
- *@result* は呼び出し側のバッチ内で宣言する必要はありません。これは、このパラメータが、mathtutor に渡されるパラメータの名前であるためです。
- *@result* の変更された値は execute 文の中で割り当てられた変数 (この場合は*@guess*) 内の呼び出し側に返されますが、見出しは *@result* がそのまま表示されます。

execute 文の後の条件付きの句で *@guess* の初期値を使用する場合は、プロシージャ・コールの間に別の変数名で @guess の初期値を保管します。次の例に、上記の 2 番目と 3 番目の項目について具体的に示します。つまり、ストアド・プロシージャの実行中に変数の値を保持する *@store* を使用し、条件句の中で *@guess* に返された「新しい」値を使用しています。

```
declare @guess int
declare @store int
select @guess = 32
select @store = @guess
execute mathtutor 5, 6,
@result = @guess output
select Your_answer = @store,
Right_answer = @guess
if @guess = @store
    print "Bingo!"
else
    print "Wrong, wrong, wrong!"

(1 row affected)
```

```
(1 row affected)
(return status = 0)

@result
-----------
         30

 Your_answer Right_answer
 ----------- ------------
          32           30

Wrong, wrong, wrong!
```

新刊本の売り上げによって作家の印税の割合が変化するかどうかを調べるストアド・プロシージャの例を次に示します (*@pc* パラメータは output パラメータとして定義されています)。

```
create proc roy_check @title tid, @newsales int,
        @pc int output
as
declare @newtotal int
select @newtotal = (select titles.total_sales + @newsales
from titles where title_id = @title)
select @pc = royalty from  roysched
   where @newtotal  >= roysched.lorange and
          @newtotal < roysched.hirange
   and roysched.title_id = @title
```

次の SQL バッチは、*percent* 変数に値を代入してから roy_check を呼び出します。リターン・パラメータは、バッチ内の次の文が実行される前に表示されます。

```
declare @percent int
select @percent = 10
execute roy_check "BU1032", 1050, @pc = @percent output
  select Percent = @percent
go

(1 row affected)
(return status = 0)

@pc
-----------
         12
Percent
-----------
         12

(1 row affected)
```

次のストアド・プロシージャは、roy_check プロシージャを呼び出し、条件付きの句の中で *percent* の戻り値を使用します。

```
create proc newsales @title tid, @newsales int
as
declare @percent int
declare @stor_pc int
select @percent = (select royalty from roysched, titles
        where roysched.title_id = @title
        and total_sales >= roysched.lorange
        and total_sales < roysched.hirange
        and roysched.title_id = titles.title_id)
select @stor_pc = @percent
execute roy_check @title, @newsales, @pc = @percent
  output
if
  @stor_pc != @percent
begin
  print "Royalty is changed."
  select Percent = @percent
end
else
  print "Royalty is the same."
```

前述のバッチで使用したものと同じパラメータを使用してこのストアド・プロシージャを実行した場合は、次のように表示されます。

```
execute newsales "BU1032", 1050

Royalty is changed
Percent
-----------
         12

(1 row affected, return status = 0)
```

roy_check を呼び出す前記の 2 つの例では、*@pc* は roy_check に渡されるパラメータで、*@percent* は出力を含む変数名です。newsales によって roy_check が実行されるとき、*@percent* に返される値は、渡される他のパラメータによって異なることがあります。*percent* に返された値を *@pc* の初期値と比較するには、この初期値を別の変数に格納します。上記の例では、その値を *stor_pc* に保存しました。

パラメータに渡される値

パラメータに値を渡すには、次の形式で行います。

@parameter = @variable

定数をそのままで渡すことはできません。戻り値を「受け取る」ための変数名が必要です。パラメータは、text、unitext、または image を除く、どのような Adaptive Server データ型でもかまいません。

注意 ストアド・プロシージャに複数のパラメータを指定する必要がある場合は、戻り値のパラメータを execute 文の最後で渡すか、または後続のすべてのパラメータを *@parameter = value* の形式で渡します。

output キーワード

ストアド・プロシージャは複数の値を返すことができますが、それぞれの値は、ストアド・プロシージャと呼び出し側の文の中で、次のように output 変数として定義する必要があります。output キーワードは、out と省略できます。

```
exec myproc @a = @myvara out, @b = @myvarb out
```

プロシージャを実行している間に output キーワードを指定し、さらに、パラメータがストアド・プロシージャ内で output キーワードを使用して定義されていない場合は、エラー・メッセージが表示されます。戻り値の指定を含んでいるプロシージャを、output キーワードを使用して戻り値を要求しないで呼び出す場合は、エラーにはなりません。ただし、戻り値は表示されません。ストアド・プロシージャの作成者はユーザがアクセスできる情報を制御でき、ユーザは自分で生成した変数を制御できます。

ストアド・プロシージャに関連する規則

ストアド・プロシージャを作成するための補足の規則を次に示します。

- 単一のバッチ内で create procedure 文を他の文と結合することはできない。
- create procedure の定義自体には、use 文と次の create 文以外のすべての種類の SQL 文をいくつでも含めることができる。
  - create view
  - create default
  - create rule
  - create trigger
  - create procedure
- プロシージャ内で他のデータベース・オブジェクトを作成できる。オブジェクトは、それを作成したプロシージャと同じプロシージャ内で参照できる。ただし、オブジェクトは、参照される前に作成されている必要がある。オブジェクトの create 文は、プロシージャ内の文の実際の順序で最初に置く必要がある。
- ストアド・プロシージャ内では、ある名前のオブジェクトを作成し、削除してから、同じ名前のオブジェクトを新しく作成することはできない。

- ストアド・プロシージャ内で定義されたオブジェクトは、プロシージャがコンパイルされるときではなく、実行されるときに Adaptive Server によって実際に作成されます。
- 別のプロシージャを呼び出すプロシージャを実行すると、呼び出されたプロシージャは、最初のプロシージャによって作成されたオブジェクトをアクセスできる。
- プロシージャ内ではテンポラリ・テーブルを参照できる。
- プロシージャ内で、# プレフィクスを使用してテンポラリ・テーブルを作成した場合、テンポラリ・テーブルは、プロシージャの実行中だけに存在する。プロシージャが終了すると、テンポラリ・テーブルは消去される。create table tempdb..*tablename* で作成したテンポラリ・テーブルは、明示的に削除しないかぎり消去されない。
- 1 つのストアド・プロシージャに指定できるパラメータの最大数は 255 である。
- プロシージャ内のローカルおよびグローバル変数の最大数は、使用できるメモリによってのみ制限されます。

プロシージャ内での名前の修飾

ストアド・プロシージャを、その所有者以外のユーザが使用する場合、ストアド・プロシージャ内の create table と dbcc で使用しているオブジェクト名は、オブジェクトの所有者で「修飾」する必要があります。ストアド・プロシージャ内で select や insert など他の文で使用されているオブジェクト名は、そのプロシージャのコンパイル時に解析されるので、修飾する必要がありません。

たとえば、ユーザ “mary” がテーブル marytab を所有しており、Mary 以外のユーザが、このテーブルが使用されているプロシージャを実行するとします。この場合、プロシージャ内の select または insert 文でテーブル名 Mary が使用されているときは、このテーブル名を Mary という名前で修飾します。オブジェクト名はプロシージャのコンパイル時に解決され、データベース ID またはオブジェクト ID のペアとして保管されます。このペアを実行時に使用できない場合、オブジェクトは再度解決されます。所有者の名前で修飾されていない場合、サーバは、ストアド・プロシージャを実行しているユーザによって所有されている marytab ではなく、ユーザ “mary” によって所有されている marytab というテーブルを検索します。オブジェクト ID “marytab” が見つからない場合は、データベース所有者によって所有される同じ名前のオブジェクトが検索されます。

したがって、marytab が修飾されず、ユーザ “john” がプロシージャを実行しようとすると、Adaptive Server は、プロシージャの所有者 (この場合は “mary”) が所有する、またはユーザ・テーブルが存在しない場合はデータベース所有者が所有する marytab というテーブルを探します。たとえば、mary.marytab テーブルが削除されると、プロシージャは dbo.marytab を参照します。

- create table で使用しているオブジェクト名をオブジェクトの所有者名で修飾できない場合は、“dbo” または “guest” を使用してオブジェクト名を修飾します。
- sa_role 権限を持つユーザがストアド・プロシージャを実行する場合、ユーザはテーブル名を tempdb.dbo.mytab として修飾する必要があります。
- sa_role 権限のないユーザがストアド・プロシージャを実行する場合、ユーザはテーブル名を tempdb.guest.mytab として修飾する必要があります。テンポラリ・データベース内のオブジェクト名がすでにデフォルトの所有者の名前で修飾されている場合、sa_role 権限のないユーザがストアド・プロシージャを実行すると、次のようなクエリは正しいオブジェクト ID を返さない可能性があります。

  ```
  select object_id ('tempdb..mytab')
  ```

  sa_role 権限がない場合に正しいオブジェクト ID を取得するには、execute コマンドを使用します。

  ```
  exec("select object_id('tempdb..mytab')")
  ```

ストアド・プロシージャの名前の変更

ストアド・プロシージャの名前を変更するには、sp_rename を使用します。

sp_rename *objname*, *newname*

たとえば、showall を countall に名前変更するには、次のようにします。

```
sp_rename showall, countall
```

新しい名前は、識別子の規則に従います。名前を変更できるのは、ユーザ自身のストアド・プロシージャだけです。データベース所有者は、すべてのユーザのストアド・プロシージャの名前を変更できます。ストアド・プロシージャは、現在のデータベース内にある必要があります。

プロシージャによって参照されるオブジェクト名の変更

プロシージャが参照するオブジェクト名を変更する場合は、プロシージャを削除して、再作成する必要があります。名前が変更されたテーブルやビューを参照するストアド・プロシージャは、しばらくの間は正常に機能するように見えるかもしれませんが、実際には、Adaptive Server が再コンパイルするときまでしか機能しません。再コンパイルは、多くの理由によってユーザへ通知されることなく行われます。

プロシージャが参照するオブジェクトのレポートを取得するには、sp_depends を使用します。

セキュリティ・メカニズムとしてのストアド・プロシージャの使用

ストアド・プロシージャをセキュリティ・メカニズムとして使用してテーブルの情報へのアクセスを制御したりデータの変更の機能を制御できます。たとえば、他のユーザが、自分の所有するテーブルで select コマンドを使用するパーミッションを拒否し、他のユーザが特定のローやカラムだけを参照できるようにするストアド・プロシージャを作成できます。update、delete、または insert 文を制限するのにも、ストアド・プロシージャを使用できます。

ストアド・プロシージャの所有者は、プロシージャで使用するテーブルとビューの所有者である必要があります。システム管理者でも、他のユーザのテーブルのパーミッションを付与されていなければ、そのテーブルでオペレーションを実行するストアド・プロシージャを作成できません。

『システム管理ガイド 第 1 巻』の「第 17 章 ユーザ・パーミッションの管理」を参照してください。

ストアド・プロシージャの削除

ストアド・プロシージャを削除するには、drop procedure を使用します。

```
drop proc[edure] [owner.]procedure_name
    [, [owner.]procedure_name] ...
```

削除されたストアド・プロシージャが別のストアド・プロシージャに呼び出された場合、Adaptive Server はエラー・メッセージを表示します。ただし、新しいプロシージャが同じ名前で定義されて、削除されたプロシージャと置き換えられた場合は、元のプロシージャを参照する他のプロシージャによって正常に呼び出されます。

プロシージャは、いったんグループ化されると、個別には削除できません。

システム・プロシージャ

システム・プロシージャには、次のような機能があります。

- システム・テーブルからすばやく情報を検索する機能
- データベース管理およびシステム・テーブルの更新を含む他のタスクを実行するためのメカニズム

ほとんどの場合、システム・テーブルはストアド・プロシージャによって更新できます。システム管理者は、設定変数を変更し、reconfigure with override コマンドを発行することによって、システム・テーブルの直接的な更新を行うことを許可します。『システム管理ガイド 第 1 巻』の「第 17 章 ユーザ・パーミッションの管理」を参照してください。

すべてのシステム・プロシージャの名前は、“sp_” で始まります。システム・プロシージャは、Adaptive Server のインストールの間に sybsystemprocs データベース内の installmaster スクリプトによって作成されます。通常、システム・プロシージャの名前はその目的を表しています。たとえば、sp_addalias はエイリアスを追加します。

システム・プロシージャの実行

システム・プロシージャは、どのデータベースからでも実行できます。sybsystemprocs データベース以外のデータベースからシステム・プロシージャが実行された場合、システム・テーブルへの参照はすべてのプロシージャが実行されているデータベースにマップされます。たとえば、pubs2 のデータベース所有者が pubs2 から sp_adduser を実行した場合、新しいユーザが pubs2..sysusers に追加されます。特定のデータベースでシステム・プロシージャを実行するには、use コマンドを使用してそのデータベースをオープンし、プロシージャを実行するか、プロシージャ名をデータベース名で修飾します。

システム・プロシージャのパラメータがオブジェクト名であり、オブジェクト名がデータベース名、または所有者名によって修飾されている場合は、名前全体を一重または二重の引用符で囲みます。

システム・プロシージャに対するパーミッション

システム・プロシージャは sybsystemprocs データベースに配置されているため、それらのパーミッションもこのデータベースに設定されます。システム・プロシージャの一部には、データベース所有者しか実行できないものがあります。このようなプロシージャは、プロシージャを実行しているユーザが、実行しているデータベースの所有者であることを確認します。

それ以外のシステム・プロシージャは、execute パーミッションを与えられたどのユーザでも実行できますが、このパーミッションは sybsystemprocs データベースで付与されている必要があります。これにより、次の 2 つの結論が得られます。

- ユーザは、すべてのデータベースにおいてシステム・プロシージャを実行するパーミッションを持つことができる場合と、どのデータベースにおいてもそれを持つことができない場合がある。
- ユーザ・データベースの所有者は、自分自身のデータベース内にあるシステム・プロシージャのパーミッションを直接制御することはできません。

システム・プロシージャのタイプ

システム・プロシージャは、監査、セキュリティ管理、データ定義などの機能別にグループ化できます。次の項では、システム・プロシージャをタイプ別に示しています。すべてのシステム・プロシージャの詳細については、『リファレンス・マニュアル：プロシージャ』を参照してください。ここでは、システム・プロシージャをアルファベット順に示しています。

Sybase が提供するその他のプロシージャ

Sybase は、カタログ・ストアド・プロシージャ、システム拡張ストアド・プロシージャ (システム ESP)、dbcc プロシージャを提供しています。

カタログ・ストアド・プロシージャ

カタログ・ストアド・プロシージャは、表形式でシステム・テーブルからの情報を検索するシステム・プロシージャです。

システム拡張ストアド・プロシージャ

拡張ストアド・プロシージャ (ESP: Extended stored procedures) は、Adaptive Server から手続き型言語関数を呼び出します。システム拡張ストアド・プロシージャはインストール時に *installmaster* によって作成され、sybsystemprocs データベースに保管されます。これは、システム管理者によって所有されます。どのデータベースからでも実行でき、名前は “xp_” で始まります。

「第 18 章 拡張ストアド・プロシージャの使用」を参照してください。

dbcc プロシージャ

dbcc プロシージャは、*installdbccdb* によって作成されます。dbcc checkstorage によって作成される情報のレポートを生成するストアド・プロシージャです。これらのプロシージャは dbccdb データベース、または代替データベースである dbccalt に存在しています。

ストアド・プロシージャに関する情報の取得

いくつかのシステム・プロシージャによって、システム・テーブルからストアド・プロシージャに関する情報を取得できます。

sp_help によるレポートの取得

sp_help を使用して、ストアド・プロシージャに関するレポートを取得できます。たとえば、以下を使用して、pubs2 データベースの一部であるストアド・プロシージャ byroyalty についての情報を取得できます。

```
sp_help byroyalty

Name       Owner   Object_type        Create_date
---------   ------  ----------------
byroyalty  dbo     stored procedure   Jul 27 2005 4:30PM

(1 row affected)
Parameter_name Type    Length   Prec  Scale  Param_order Mode
-------------- ------  ------   ----  -----  -----------
@percentage    int          4   NULL   NULL            1

(return status = 0)
```

sybsystemprocs データベースを使用しているときに sp_help を実行すると、システム・プロシージャについてのヘルプ情報を取得できます。

sp_helptext によるプロシージャのソース・テキストの表示

create procedure 文のテキストを表示するには、sp_helptext を実行します。

```
sp_helptext byroyalty

# Lines of Text
---------------
              1

(1 row affected)

text
--------------------------------------------------
create procedure byroyalty @percentage int
as
select au_id from titleauthor
where titleauthor.royaltyper = @percentage

(1 row affected, return status = 0)
```

sybsystemprocs データベースを使用しているときに sp_helptext を実行すると、システム・プロシージャのソース・テキストを表示できます。

ストアド・プロシージャのソース・テキストを sp_hidetext を使用して暗号化した場合、Adaptive Server はテキストが隠されたことを伝えるメッセージを表示します。『リファレンス・マニュアル：プロシージャ』を参照してください。

sp_depends による従属オブジェクトの識別

sp_depends によって、指定するオブジェクトを参照するすべてのストアド・プロシージャ、または指定するオブジェクトが依存するすべてのプロシージャがリストされます。

たとえば、次のコマンドは、ユーザが作成したストアド・プロシージャ byroyalty によって参照されるすべてのオブジェクトを、次のようにリストします。

```
sp_depends byroyalty

Things the object references in the current database.
object           type       updated      selected
---------------- ---------- ---------    --------
dbo.titleauthor  user table  no          no

(return status = 0)
```

次の文は、sp_depends を使用して、titleauthor テーブルを参照するすべてのオブジェクトをリストします。

```
sp_depends titleauthor

Things inside the current database that reference the object.

object          type
--------------  ------------------
dbo.byroyalty   stored procedure
dbo.titleview   view

(return status = 0)

Dependent objects that reference all columns in the table.Use
sp_depends on each column to get more information.Columns
referenced in stored procedures views, or triggers are not
included in this report.
........
(1 row affected)
(return status = 0)
```

プロシージャが参照するオブジェクト名が変更された場合は、プロシージャを削除して、再作成する必要があります。

deferred_name_resolution を指定した *sp_depends* の使用

遅延名前解決依存情報を使用して作成されるプロシージャは、実行時に作成されます。したがって、遅延名前解決を使用して作成され、まだ実行されていないプロシージャが sp_depends によって実行されるときにメッセージが表示されます。

```
sp_depends p
----------
The dependencies of the stored procedure cannot be determined
until the first successful execution.
(return status = 0)
```

実行が最初に成功すると、依存情報が作成され、sp_depends の実行によって想定された情報が返されます。

次に例を示します。

```
set deferred_name_resolution on
-----------------

create procedure p as
select id from sysobjects
where id =

1sp_depends p
------------
The dependencies of the stored procedure cannot be determined
until the first successful execution.

(return status = 0)

exec p
id

 -----------
           1


(1 row affected)
(return status = 0)


sp_depends p
----------------

The object references in the current database.
object                         type
updated
         selected

------------------------------ ------------------------ -------
-------
         ----------------

 dbo.sysobjects                            system table
           no
                      no

(return status = 0)
```

sp_helprotect によるパーミッションの識別

sp_helprotect は、ストアド・プロシージャ (または他のデータベース・オブジェクト) のパーミッションをレポートします。次に例を示します。

```
sp_helprotect byroyalty

grantor    grantee    type  action    object    column  grantable
---------  ---------  ----  --------- -------   -----   ---------
dbo        public     Grant Execute   byroyalty All     FALSE

(return status = 0)
```

第 18 章

拡張ストアド・プロシージャの使用

「拡張ストアド・プロシージャ」(ESP) は、Adaptive Server 内から外部手続き型言語関数を呼び出すためのメカニズムを提供します。ユーザはストアド・プロシージャと同じ構文を使用して ESP を呼び出します。違いは、ESP は Transact-SQL 文ではなく、手続き型言語コードを実行する点です。

概要

拡張ストアド・プロシージャを使用することで、Adaptive Serverから外部手続き型言語関数を動的にロードして実行できます。各 ESP は、対応する関数に関連付けられており、Adaptive Server から ESP を呼び出すと、それに対応する関数が実行されます。

ESP によって、Adaptive Server は、Adaptive Server 内で発生しているイベントに応じて Adaptive Server の外部のタスクを実行できます。たとえば、証券を売るための ESP 関数を作成できます。この ESP は、証券の価格がある値に達したときに起動されるトリガに応じて呼び出されます。そのほかに、リレーショナル・データベース・システム内で発生しているイベントに応じて、電子メールの通知やネットワーク規模の通信を送る ESP 関数を作成できます。

ESP において、「手続き型言語」とは、C 言語の呼び出し機能と C 言語データ型の操作機能を持つプログラミング言語を意味します。

関数を ESP としてデータベースに登録すると、ストアド・プロシージャのように isql、トリガ、別のストアド・プロシージャ、またはクライアント・アプリケーションから ESP を呼び出せます。

ESP には、次のような機能があります。

- 入力パラメータを使用できる
- 成功または失敗を示すステータス値と、失敗した場合はその理由を戻す
- 出力パラメータの値を戻す
- 結果セットを戻す

Adaptive Server はシステム ESP をいくつか提供します。たとえば、xp_cmdshell というシステム ESP を使用して Adaptive Server 内からオペレーティング・システム・コマンドを実行できます。また、Open Server アプリケーション・プログラミング・インタフェース (API) のサブセットを使用して、独自の ESP を作成できます。

XP Server

ESP は、Adaptive Server と同じマシン上で稼働する、XP Server と呼ばれる Open Server アプリケーションによって実装されます。Adaptive Server と XP Server は、RPC (リモート・プロシージャ・コール) を介して通信します。別のプロセスで ESP を実行することで、ESP コードの誤りが原因で発生する障害が Adaptive Server に影響を及ぼすのを防げます。RPC より ESP を使用する利点は、ストアド・プロシージャを実行するのと同様に ESP を Adaptive Server で実行できることです。ESP を実行するだけなら、Open Server は必要ありません。

XP Server は、Adaptive Server とともに自動的にインストールされます。ただし、XP Server ライブラリを開発する場合は、Open Server ライセンスを購入してください。XP Server の DLL を使用して XP Server のコマンドを実行するために必要なものはすべて、Adaptive Server ライセンスに含まれています。

Adaptive Server が ESP を実行するためには、XP Server が実行されている必要があります。Adaptive Server が ESP の初回起動時に XP Server を起動し、Adaptive Server の終了時に XP Server をシャットダウンします。

Windows では、start mail session 設定パラメータを 1 に設定した場合、Adaptive Server の起動時に XP Server も自動的に起動します。

CIS RPC メカニズムの使用

XP Server プロシージャは、サイト・ハンドラによるルート指定以外にも、CIS RPC メカニズムを使用して実行できます。このメカニズムを使用するために必要なオプションの cis rpc handling と negotiated logins を設定するには、次のように入力します。

```
//to set 'cis rpc handling'//
sp_configure 'cis rpc handling', 1
//or at the session level//
set 'cis rpc handling' on
```

```
//to set 'negotiated logins'//
sp_serveroption XPServername, 'negotiated logins', true
```

どちらかのオプションが設定されていない場合、ESP はサイト・ハンドラによってルート指定され、警告メッセージは表示されません。

sybesp_dll_version の使用

XP Server にロードされるすべてのライブラリで関数 sybesp_dll_version() を実装することをおすすめします。この関数は、DLL によって使用される Open Server API バージョンを返します。次のように入力します。

```
CS_INT sybesp_dll_version()
----------
CS_CURRENT_VERSION
```

CS_CURRENT_VERSION は、Open Server API で定義されているマクロです。DLL の使用によって、特定の値をハードコードする手間を防ぎます。CS_CURRENT_VERSION が実装されていない場合、XP Server はバージョンの照合を行いません。ログ・ファイルには、エラー・メッセージ #11554 が書き込まれます (エラー・メッセージについては、『トラブルシューティングおよびエラー・メッセージ・ガイド』を参照してください)。しかし、XP Server は継続して DLL をロードします。

バージョンの不一致がある場合、XP Server はログ・ファイルにエラー メッセージ 11555 を書き込みますが、DLL のロードを続行します。

XP Server の手動による起動

通常は、ユーザが XP Server を手動で起動する必要はありません。XP Server は、セッション最初の ESP 要求を受信した Adaptive Server によって起動されるからです。ただし、独自の ESP を作成、デバッグしている最中に、xpserver ユーティリティを使用し、コマンド・ラインから XP Server を手動で起動する必要が生じることもあります。xpserver の構文については、『ユーティリティ・ガイド』を参照してください。

ダイナミック・リンク・ライブラリ・サポート

ESP コードを含む手続き型関数は、コンパイルされ、ダイナミック・リンク・ライブラリ (DLL) にリンクされます。DLL は、ESP の実行要求に応じて XP Server メモリにロードされます。ライブラリは、次のいずれかが起こるまでロードされたままになります。

- XP Server の終了
- sp_freedll システム・プロシージャの呼び出し
- sp_configure を使用して、esp unload dll 設定パラメータが設定される

Open Server API

Adaptive Server は Open Server API を使用します。この API によって、ユーザは Adaptive Server が提供するシステム ESP を実行できます。また、Open Server API を使用して、独自の ESP を実装することもできます。

表 18-1 は、ESP の開発に必要な Open Server ルーチンのリストです。これらのルーチンの詳細については、『Open Server Server-Library/C リファレンス・マニュアル』を参照してください。

表 18-1: ESP サポートのための Open Server ルーチン

関数	目的
srv_bind	プログラム変数を記述し、パラメータにバインドする。
srv_descfmt	パラメータの記述。
srv_numparams	ESP クライアント要求内のパラメータ数を返す。
srv_senddone	結果完了メッセージの送信。
srv_sendinfo	メッセージの送信。
srv_sendstatus	ステータス値の送信。
srv_xferdata	パラメータまたはデータの送信と受信。
srv_yield	現在のスレッドの実行を中断し、別のスレッドの実行を許可する。

ESP 関数をコーディングおよびコンパイルし、DLL にリンクしたあと、次のように create procedure コマンドで as external name 句を使用して、関数の ESP を作成できます。

```
create procedure procedure_name [parameter_list]
    as external name dll_name
```

procedure_name は、ESP の名前です。ESP 名には、DLL 内の対応する関数名と同じ名前を指定する必要があります。ESP はデータベース・オブジェクトなので、識別子の規則に従って命名します。

dll_name は、実装する関数を格納する DLL 名です。DLL の命名規則は、プラットフォームによって異なります。

表 18-2: DDL 拡張子に関する命名規則

プラットフォーム	DLL 拡張子
HP 9000/800 HP-UX	*.sl*
Sun Solaris	*.so*
Windows	*.dll*

次の文は、getmsgs という名前の ESP を作成し、*msgs.dll* に格納します。getmsgs ESP にパラメータはありません。次に、Adaptive Server Windows 版の例を示します。

```
create procedure getmsgs
as external name "msgs.dll"
```

次の文は、getonemsg という名前の ESP を作成し、*msgs.dll* に格納します。getonemsg ESP は、メッセージ番号を単一のパラメータとして受け付けます。

```
create procedure getonemsg @msg int
as external name "msgs.dll"
```

Adaptive Server は、ESP を作成すると、オブジェクト・タイプ “XP” および syscomments システム・テーブルの text カラムにその ESP の関数がある DLL の名前とともに、そのプロシージャ名を sysobjects システム・テーブルに格納します。

ユーザ定義のストアド・プロシージャやシステム・プロシージャを実行する要領で、ESP を実行します。キーワード execute とストアド・プロシージャ名を使用できます。または、そのプロシージャを単独で Adaptive Server に実行する場合や、そのプロシージャがバッチ内の最初の文である場合は、プロシージャ名を指定するだけです。たとえば、次のいずれかの方法で getmsgs を実行できます。

```
getmsgs
execute getmsgs
exec getmsgs
```

次のいずれかの方法で、getonemsg を実行できます。

```
getonemsg 20
getonemsg @msg=20
execute getonemsg 20
execute getonemsg @msg=20
exec getonemsg 20
exec getonemsg @msg=20
```

ESP とパーミッション

通常のストアド・プロシージャと同じように、ESP のパーミッションを付与したり取り消したりできます。

通常の Adaptive Server セキュリティに加え、xp_cmdshell context 設定パラメータを使用して、xp_cmdshell システム ESP の実行パーミッションをシステム管理権限を持つユーザだけに制限できます。この設定パラメータを使用することで、xp_cmdshell を使用してコマンド行から直接実行することを許可されていない一般ユーザが、オペレーティング・システム・コマンドを実行するのを防ぎます。xp_cmdshell 設定パラメータの動作は、プラットフォーム固有です。

デフォルトでは、ユーザが xp_cmdshell を実行するには、sa_role が必要です。xp_cmdshell を使用するためのパーミッションを他のユーザに付与するには、grant コマンドを使用します。パーミッションを取り消すには、revoke を使用します。grant または revoke パーミッションは、xp_cmdshell のコンテキストが 0 または 1 のどちらに設定されていても適用できます。

ESP とパフォーマンス

Adaptive Server と XP Server は両方とも同じマシンに常駐するので、XP Server が大量のリソースを消費する関数を実行する場合は、互いのパフォーマンスに影響を及ぼす可能性があります。

esp execution priority を使用して、XP Server のスレッドの優先度を高く設定できます。これによって、Open Server スケジューラは、その XP Server スレッドを実行キュー内の他のスレッドより前に実行します。反対に、XP Server スレッドの優先度を低く設定すると、スケジューラは、他に実行するスレッドがない場合にだけ XP Server スレッドを実行します。esp execution priority のデフォルト値は 8 ですが、0 ～ 15 の値を設定できます。

同じサーバ上で実行するすべての ESP は、互いに実行を譲り合う必要があります。これには、XP Server スレッドを中断してから別のスレッドに実行を許可する、Open Server の srv_yield ルーチンを使用します。

詳細については、『Open Server Server-Library/C リファレンス・マニュアル』のマルチスレッド・プログラミングに関する章を参照してください。

XP Server のメモリ容量を最小限に抑えるには、DLL をロードした ESP 要求が終了した後、XP Server メモリから DLL をアンロードします。このためには、esp unload dll を設定します。このパラメータを設定することで、ESP の実行が終了後、DLL が自動的にアンロードされます。esp unload dll が設定されていない場合は、sp_freedll を使用して DLL を明示的に解放します。

システム ESP をサポートする DLL は、アンロードできません。

ESP 用の関数の作成

ESP を実装する関数の内容について、制限はありません。ただし、次の機能を持つ手続き型プログラミング言語で作成する必要があります。

- C 言語関数の呼び出し
- C 言語データ型の操作
- Open Server API とのリンク機能

例外として、ESP 関数は、Windows 上では C 言語のランタイム・シグナル・ルーチンを呼び出せません。Open Server は、Windows 上でシグナル処理をサポートしないので、このような呼び出しを行うと XP Server が失敗する可能性があります。

Open Client API を使用することで、ESP 関数は、その関数の呼び出し元の Adaptive Server か別の Adaptive Server に要求を送信できます。

ESP 開発用ファイル

開発用の Open Server ライブラリを使用するための Open Server ライセンスをまず購入してください。ESP 開発に必要なヘッダ・ファイルは、*$SYBASE/$SYBASE_OCS/include* にあります。これらのファイルをソース・ファイルに入れるには、次のファイルをソース・コード内に指定します。

- *ospublic.h*
- *oserror.h*

Open Server ライブラリは *$SYBASE/$SYBASE_OCS/lib* にあります。「ESP 関数の例」(556 ページ)で示すサンプル・プログラムのソースは、*$SYBASE/$SYBASE_ASE/sample/esp* にあります。

Open Server データ構造体

ESP 関数を作成するのに、これらのデータ構造体を使用すると便利です。

- SRV_PROC － すべての ESP 関数は、SRV_PROC 構造体へのポインタとなる単一パラメータを受け入れるようにコーディングされます。SRV_PROC 構造体は、関数とそれを呼び出したプロセスの間で情報の引き渡しを行います。ESP 開発者は、この構造体を直接操作できません。

  ESP 関数は、パラメータ・タイプとデータを取得し、出力パラメータ、ステータス・コード、結果セットを返す Open Server ルーチンに、SRV_PROC ポインタを渡します。

- CS_SERVERMSG － Open Server では、CS_SERVERMSG 構造体を使用し、*srv_sendinfo* ルーチンを介してクライアントにエラー・メッセージを送信します。CS_SERVERMSG については、『Open Server-Library/C リファレンス・マニュアル』を参照してください。
- CS_DATAFMT － Open Server は、CS_DATAFMT 構造体を使用して、データ値とプログラミング変数を記述します。

Open Server のリターン・コード

Open Server 関数は、CS_RETCODE タイプのコードを返します。ESP 関数の最も一般的な CS_RETCODE 値は、次のとおりです。

- CS_SUCCEED
- CS_FAIL

ESP 関数の基本構造

ESP 関数が Open Server API と対話する場合の基本構造は、次のようになります。

1 パラメータ数を取得する。

2 入出力パラメータの値を取得し、それをローカル変数にバインドする。

3 入力パラメータの値を使用して処理を実行し、結果をローカル変数に格納する。

4 出力パラメータを適切な値で初期化し、それをローカル変数とバインドし、クライアントに転送する。

5 srv_sendinfo() を使用して返されたローをクライアントに送る。

6 srv_sendstatus() を使用してステータスをクライアントに送る。

7 srv_senddone を使用して処理が終了したことをクライアントに知らせる。

8 エラー状態の場合は、srv_sendinfo を使用してエラー・メッセージをクライアントに送る。

Open Server ルーチンの詳細については『Open Server Server-Library/C リファレンス・マニュアル』を参照してください。

ESP 関数の例

xp_echo.c には、ユーザが指定する入力パラメータを受け取り、ESP を呼び出す ESP クライアントにそのパラメータをエコーする ESP が含まれています。以下に示す例にはメッセージとステータスを送信する xp_message 関数と、入力パラメータを処理し、エコーを実行する xp_echo 関数が含まれています。このサンプルをテンプレートとして使用して、独自の ESP 関数を構築できます。このサンプルのソースは、*$SYBASE/$SYBASE_ASE/sample/esp* にあります。

```
/*
** xp_echo.c
**
**      Description:
**              The following sample program is generic in
**              nature.It echoes an input string which is
**              passed as the first parameter to the xp_echo
**              ESP. This string is retrieved into a buffer
**              and then sent back (echoed) to the ESP client.
*/
#include <string.h>
#include <stdlib.h>
#include <malloc.h>
/* Required Open Server include files.*/
```

```
#include <ospublic.h>
#include <oserror.h>
/*
** Constant defining the length of the buffer that receives the
** input string.All of the Adaptive Server parameters related
** to ESP may not exceed 255 char long.
*/
#define ECHO_BUF_LEN    255
/*
** Function:
**      xp_message
**      Purpose: Sends information, status and completion of the
**      command to the server.
** Input:
**      SRV_PROC *
**      char *  a message string.
** Output:
**      void
*/
void xp_message
(
        SRV_PROC *srvproc,  /* Pointer to Open Server thread
                            control structure */
        char     *message_string /* Input message string */
)
{
        /*
        ** Declare a variable that will contain information
        ** about the message being sent to the SQL client.
        */
        CS_SERVERMSG    *errmsgp;
        /*
        ** A SRV_DONE_MORE instead of a SRV_DONE_FINAL must
        ** complete the result set of an Extended Stored
        ** Procedure.
        */
        srv_senddone(srvproc, SRV_DONE_MORE, 0, 0);
        free(errmsgp);
}
/*
** Function: xp_echo
** Purpose:
**      Given an input string, this string is echoed as an output
**      string to the corresponding SQL (ESP) client.
** Input:
**      SRV_PROC *
** Output
**      SUCCESS or FAILURE
*/
CS_RETCODE xp_echo
(
```

```
        SRV_PROC        *srvproc
)
{
        CS_INT          paramnum;  /* number of parameters */
        CS_CHAR         echo_str_buf[ECHO_BUF_LEN + 1];
                               /* buffer to hold input string */
        CS_RETCODE      result = CS_SUCCEED;
        CS_DATAFMT      paramfmt; /* input/output param format */
        CS_INT          len;      /* Length of input param */
        CS_SMALLINT     outlen;
        /*
        ** Get number of input parameters.*/
        */


        srv_numparams(srvproc, &paramnum);
        /*
        ** Only one parameter is expected.*/
        */
        if (paramnum != 1)
        {
                /*
                ** Send a usage error message.*/
                */
                xp_message(srvproc, "Invalid number of
                 parameters");
                  result = CS_FAIL;
        }
      else
        {
                /*
                ** Perform initializations.
                */
                  outlen = CS_GOODDATA;
                memset(&paramfmt, (CS_INT)0,
                    (CS_INT)sizeof(CS_DATAFMT));
                /*
                ** We are receiving data through an ESP as the
                ** first parameter.So describe this expected
                ** parameter.
                */
                if ((result == CS_SUCCEED) &&
                  srv_descfmt(srvproc, CS_GET
                  SRV_RPCDATA, 1, &paramfmt) != CS_SUCCEED)
                {
                          result = CS_FAIL;
                }
                /*
                ** Describe and bind the buffer to receive the
                ** parameter.
                */
```

```
if ((result == CS_SUCCEED) &&
  (srv_bind(srvproc, CS_GET, SRV_RPCDATA,
   1, &paramfmt,(CS_BYTE *) echo_str_buf,
   &len, &outlen) != CS_SUCCEED))
{
          result = CS_FAIL;
}
/* Receive the expected data.*/
if ((result == CS_SUCCEED) &&
  srv_xferdata(srvproc,CS_GET,SRV_RPCDATA)


  != CS_SUCCEED)
{
          result = CS_FAIL;
}               /*
** Now we have the input info and are ready to
** send the output info.
*/
if (result == CS_SUCCEED)
{
        /*
        ** Perform initialization.
        */
        if (len == 0)
                outlen = CS_NULLDATA;
        else
                outlen = CS_GOODDATA;


        memset(&paramfmt, (CS_INT)0,
          (CS_INT)sizeof(CS_DATAFMT));
        strcpy(paramfmt.name, "xp_echo");
        paramfmt.namelen = CS_NULLTERM;
        paramfmt.datatype = CS_CHAR_TYPE;
        paramfmt.format = CS_FMT_NULLTERM;
        paramfmt.maxlength = ECHO_BUF_LEN;
        paramfmt.locale = (CS_LOCALE *) NULL;
        paramfmt.status |= CS_CANBENULL;
        /*
        ** Describe the data being sent.
        */
        if ((result == CS_SUCCEED) &&
                srv_descfmt(srvproc, CS_SET,
                  SRV_ROWDATA, 1, &paramfmt)
                  != CS_SUCCEED)
        {
                result = CS_FAIL;
        }
        /*
        ** Describe and bind the buffer that
        ** contains the data to be sent.
```

```
                        */
                        if ((result == CS_SUCCEED) &&
                                (srv_bind(srvproc, CS_SET,
                                SRV_ROWDATA, 1,
                                &paramfmt, (CS_BYTE *)
                                echo_str_buf, &len, &outlen)
                                != CS_SUCCEED))
                        {
                                result = CS_FAIL;
                        }
                        /*
                        ** Send the actual data.
                        */
                        if ((result == CS_SUCCEED) &&
                                srv_xferdata(srvproc, CS_SET,
                                SRV_ROWDATA)!= CS_SUCCEED)
                        {
                                result = CS_FAIL;
                        }
                }
                /*
                ** Indicate to the ESP client how the
                ** transaction was performed.
                */
                if (result == CS_FAIL)
                        srv_sendstatus(srvproc, 1);
                else
                        srv_sendstatus(srvproc, 0);
                /*
                ** Send a count of the number of rows sent to
                ** the client.
                */
                srv_senddone(srvproc,(SRV_DONE_COUNT |
                  SRV_DONE_MORE), 0, 1);
        }
        return result;
}
```

DLL の構築

必要な DLL を生成できるコンパイラであればどれでも、サーバ・プラットフォームで使用できます。

Open Server API を使用する関数のコンパイルおよびリンク方法については、『Open Client/Server プログラマーズ・ガイド補足』を参照してください。

DLL の検索順序

Windows は、次の順序で DLL を検索します。

1　アプリケーションが呼び出されたディレクトリ

2　現在のディレクトリ

3　システム・ディレクトリ (System32)

4　PATH 環境変数に指定されているディレクトリ

UNIX の場合、LD_LIBRARY_PATH 環境変数 (Solaris 上)、SHLIB_PATH (HP 上)、または AIX の LIBPATH に指定されたディレクトリ内で、指定されたディレクトリ順にライブラリを検索します。

検索パス内で ESP 関数用のライブラリが見つからない場合、XP Server は Windows 上では *$SYBASE/DLL* から、他のプラットフォームでは *$SYBASE/lib* から、ライブラリをロードしようとします。

DLL の絶対パス名は、サポートされていません。

makefile のサンプル (UNIX)

次に示す makefile の *make.unix* を使用すると、xp_echo プログラム用に動的にリンクされた共有ライブラリが UNIX プラットフォーム上に作成されます。この makefile によって、Solaris では *examples.so* ファイル、HP では *examples.sl* ファイルが生成されます。ソースは *$SYBASE/$SYBASE_ASE/sample/esp* にあるので、それを修正して独自の makefile として使用できます。

この makefile の例を使用してライブラリのサンプルを構築するには、次のように入力します。

```
make -f make.unix
```

```
#
# This makefile creates a shared library.It needs the open
# server header
# files usually installed in $SYBASE/include directory.
# This make file can be used for generating the template ESPs.
# It references the following macros:
#
# PROGRAM is the name of the shared library you may want to create.PROGRAM        =
example.so
BINARY        = $(PROGRAM)
EXAMPLEDLL    = $(PROGRAM)

# Include path where ospublic.h etc reside.You may have them in
# the standard places like /usr/lib etc.

INCLUDEPATH    = $(SYBASE)/include
```

```
# Place where the shared library will be generated.
DLLDIR        = .

RM        = /usr/bin/rm
ECHO        = echo
MODE        = normal

# Directory where the source code is kept.
SRCDIR        = .

# Where the objects will be generated.
OBJECTDIR    = .

OBJS        = xp_echo.o

CFLAGS        = -I$(INCLUDEPATH)
LDFLAGS        =  $(GLDFLAGS) -Bdynamic

DLLLDFLAGS    =  -dy -G
#=====================================================================

$(EXAMPLEDLL) : $(OBJS)
    -@$(RM) -f $(DLLDIR)/$(EXAMPLEDLL)
    -@$(ECHO) "Loading $(EXAMPLEDLL)"
    -@$(ECHO) " "
    -@$(ECHO) "    MODE:            $(MODE)"
    -@$(ECHO) "    OBJS:            $(OBJS)"
    -@$(ECHO) "    DEBUGOBJS:       $(DEBUGOBJS)"
    -@$(ECHO) " "

    cd $(OBJECTDIR); ¥
    ld -o $(DLLDIR)/$(EXAMPLEDLL) $(DEBUGOBJS) $(DLLLDFLAGS) $(OBJS)
    -@$(ECHO) "$(EXAMPLEDLL) done"
    exit 0

#=====================================================================
$(OBJS) : $(SRCDIR)/xp_echo.c
    cd $(SRCDIR); ¥
    $(CC) $(CFLAGS) -o $(OBJECTDIR)/$(OBJS) -c xp_echo.c
```

定義ファイルのサンプル

次に示す *xp_echo.def* ファイルは、*xp_echo.mak* と同じディレクトリに格納する必要があります。このファイルの EXPORTS セクションには、ESP 関数として使用されるすべての関数のリストがあります。

```
LIBRARY   examples

CODE      PRELOAD MOVEABLE DISCARDABLE
DATA      PRELOAD SINGLE
```

```
EXPORTS
        xp_echo   .   1
```

ESP の登録

ESP を作成し、それを DLL にリンクすると、その関数は ESP としてデータベースに登録されます。これによって、ユーザはその関数を ESP として実行できます。

ESP を登録するには、次のいずれかを使用します。

- Transact-SQL の create procedure コマンド
- sp_addextendedproc

create procedure の使用

create procedure の構文は次のとおりです。

```
create procedure [owner.]procedure_name
    [[(]@parameter_name datatype [= default] [output]
    [, @parameter_name datatype [= default]
    [output]]...[)]] [with recompile]
    as external name dll_name
```

procedure_name は、データベース内で認識されている ESP の名前です。この名前は、その ESP をサポートする関数名と同じにする必要があります。

パラメータの宣言は、「第 17 章 ストアド・プロシージャの使用」で説明されているストアド・プロシージャのパラメータ宣言と同じです。

with recompile 句が、ESP を作成する create procedure コマンドに組み込まれている場合、Adaptive Server はその句を無視します。

dll_name は、ESP をサポートする関数が組み込まれているライブラリ名です。拡張子が付かない名前 (*msgs*) か、またはプラットフォーム固有の拡張子が付いた名前 (Windows NT では *msgs.dll*、Solaris では *msgs.so* など) を指定します。いずれの場合も、プラットフォーム固有の拡張子は、ライブラリの実際のファイル名の一部とみなされます。

create procedure によって ESP は特定のデータベースに登録されるため、コマンドを呼び出す前に、ESP を登録するデータベースを指定します。登録先のデータベースで現在作業していない場合は、isql から use *database* コマンドを使用してデータベースを指定します。

次の文は、「ESP 関数の例」(556 ページ) の例にある xp_echo ルーチンがサポートする ESP を登録します。関数は、*examples.dll* という名前の DLL でコンパイルすると仮定します。ESP の登録先は、pubs2 データベースです。

```
use pubs2
create procedure xp_echo @in varchar(255)
as external name "examples.dll"
```

『リファレンス・マニュアル：コマンド』を参照してください。

sp_addextendedproc の使用

sp_addextendedproc は、Microsoft SQL Server で使用される構文と互換性があります。これは、create procedure の代わりとして使用できます。構文は次のとおりです。

sp_addextendedproc *esp_name*, *dll_name*

esp_name は、ESP の名前です。ESP 名は、ESP をサポートする関数の名前と一致する必要があります。

dll_name は、ESP をサポートする関数が組み込まれている DLL 名です。

sp_addextendedproc は、sa_role を持つユーザが master データベースで実行する必要があります。したがって、sp_addextendedproc は、ESP を常に master データベースに登録します。これは、現在のデータベースに ESP を登録する create procedure とは異なります。create procedure とは異なり、sp_addextendedproc は Adaptive Server でパラメータやパラメータのデフォルト値の検査ができません。

次の文は、「ESP 関数の例」(556 ページ) の例にある xp_echo ルーチンがサポートする ESP を master データベースに登録します。関数は、*examples.dll* という名前の DLL でコンパイルすると仮定します。

```
use master
sp_addextendedproc "xp_echo", "examples.dll"
```

『リファレンス・マニュアル：プロシージャ』を参照してください。

ESP の削除

ESP をデータベースから削除するには、drop procedure または sp_dropextendedproc の 2 つのコマンドを使用できます。

drop procedure の構文は、ストアド・プロシージャの構文と同じです。

drop procedure [*owner*.]*procedure_name*

次に例を示します。

```
drop procedure xp_echo
```

sp_dropextendedproc の構文は、次のとおりです。

sp_dropextendedproc *esp_name*

次に例を示します。

```
sp_dropextendedproc xp_echo
```

この 2 つの方法は両方とも、sysobjects と syscomments システム・テーブル内の ESP へのリファレンスを削除して、データベースから ESP を削除します。これらの方法は、基本となる DLL には影響を与えません。

ESP の名前の変更

ESP の名前は対応する関数名にバインドされているため、ストアド・プロシージャのように sp_rename を使用して変更することはできません。ESP 名を変更するには、次の手順に従います。

1　drop procedure または sp_dropextendedproc を使用して、ESP を削除します。

2　サポートする関数の名前を変更して、再コンパイルします。

3　create procedure または sp_addextendedproc を使用して、新しい名前で ESP を再作成します。

ESP の実行

ESP は、通常のストアド・プロシージャを実行するときと同じ execute コマンドを使用して実行します。「ストアド・プロシージャの作成と実行」(516 ページ) を参照してください。

ESP は、リモートで実行することもできます。「プロシージャのリモート実行」(530 ページ) を参照してください。

ESP の実行には、Adaptive Server と XP Server 間のリモート・プロシージャ・コールが関係するため、名前で指定されたパラメータおよび値で指定されたパラメータの両方を同じ execute コマンドに指定することはできません。すべてのパラメータは、名前または値のいずれかに統一して渡す必要があります。この点についてのみ、拡張ストアド・プロシージャの実行と通常のストアド・プロシージャの実行は異なります。

ESP は、次の値を返します。

- 成功または失敗を示すステータス値と、失敗した場合はその理由
- 出力パラメータの値
- 結果セット

ESP 関数は、Open Server ルーチン srv_sendstatus によってリターン・ステータス値をレポートします。srv_sendstatus からのリターン・ステータス値は、アプリケーション固有の値です。ただし、ゼロのステータスは、要求が正常に実行されたことを示します。

拡張ストアド・プロシージャに対するパラメータ宣言リストがない場合、Adaptive Server は指定されたパラメータをすべて無視しますが、エラー・メッセージは発行しません。宣言リストで宣言した数より多くのパラメータを ESP の実行時に指定した場合でも、Adaptive Server は指定されたパラメータを呼び出します。どのパラメータが呼び出されたかについての混乱を避けるためには、宣言リストにあるパラメータが実行時に指定されたパラメータと一致することを確認してください。また、Open Server 関数の構築時に、指定した ESP 内のパラメータ数も確認してください。

ESP 関数は、Open Server ルーチン srv_descfmt、srv_bind、および srv_xferdata を使用して、出力パラメータと結果セットの値を戻します。ESP 関数からの値の渡し方については、「ESP 関数の例」(556 ページ) と『Open Server Server-Library/C リファレンス・マニュアル』を参照してください。Adaptive Server 側では、ESP から返された値は、通常のストアド・プロシージャから返された値と同じように処理されます。

システム ESP

xp_cmdshell の他にも、Adaptive Server を Windows イベント・ログやメール・システムへ統合するなどの Windows の機能をサポートするシステム ESP がいくつかあります。

システム ESP の名前は、必ず “xp” で始まります。システム ESP は、Adaptive Server のインストール時に sybsystemprocs データベースに作成されます。システム ESP が sybsystemprocs データベース内に格納されるため、そのパーミッションも同じデータベースに格納されます。ただし、どのデータベースからでもシステム ESP を実行できます。システム ESP には、次のような種類があります。

- xp_cmdshell
- xp_deletemail
- xp_enumgroups
- xp_findnextmsg
- xp_logevent
- xp_readmail
- xp_sendmail

- xp_startmail
- xp_stopmail

システム ESP の詳細については、『リファレンス・マニュアル：プロシージャ』の「第 3 章 システム拡張ストアド・プロシージャ」を参照してください。また、『設定ガイド Windows 版』では、イベント・ログの統合など、Windows 固有の機能について詳しく説明しています。

ESP に関する情報の取得

現在のデータベースに登録されている ESP の情報を取得するには、sp_helpextendedproc を使用します。

パラメータを指定しない場合、sp_helpextendedproc はデータベース内のすべての ESP 名とそれに関連する関数を含む DLL 名を表示します。パラメータとして ESP 名を指定すると、指定した ESP の同じ情報だけが提供されます。

```
sp_helpextendedproc getmsgs

ESP Name    DLL
--------    ---------
getmsgs     msgs.dll
```

システム ESP は sybsystemprocs データベース内にあるので、ESP 名と DLL 名を表示するには、sybsystemprocs データベースを使用する必要があります。

```
use sybsystemprocs
sp_helpextendedproc

ESP Name      DLL
--------      ---------
xp_freedll    sybyesp
xp_cmdshell   sybyesp
```

ESP のソース・テキストを sp_hidetext を使用して暗号化した場合、Adaptive Server はテキストが隠されたことを伝えるメッセージを表示します。『リファレンス・マニュアル：プロシージャ』を参照してください。

ESP の例外とメッセージ

Adaptive Server は、XP Server からのすべてのメッセージと例外を処理します。Adaptive Server は通常の ESP メッセージをログ・ファイルにログするだけでなく、そのメッセージをクライアントに送信します。ユーザ定義 ESP からのユーザ定義メッセージはログされませんが、クライアントには送信されます。

ESP 関連のメッセージは、XP Server や、ESP を作成または操作するシステム・プロシージャ、またはシステム ESP によって生成される場合があります。ESP 関連のメッセージのリストについては、『トラブルシューティングおよびエラー・メッセージ・ガイド』を参照してください

ユーザ定義 ESP の関数は、Open Server ルーチン srv_sendinfo を使用してメッセージを生成できます。詳細については、「ESP 関数の例」(556 ページ) の xp_message 関数の例を参照してください。

第 19 章

カーソル：データのアクセス

「カーソル」を使うと、SQL select 文の結果のローを一度に 1 行または複数行アクセスできます。カーソルを使用すると、個々のローやひとまとまりのローを修正したり削除したりできます。

カーソル機能は、Adaptive Server バージョン 15.0 で大幅に変更されました。このバージョンではスクロール可能な読み込み専用カーソルが導入されています。スクロール可能カーソルでは、1 つ以上のローを選択でき、またロー間を前後にスクロールできます。スクロール可能カーソルは常に読み込み専用です。

非スクロール可能カーソルを使用すると、既に選択したローに戻ることはできません。また、一度に複数のローを移動することはできません。

デフォルトの前方専用カーソルも引き続きサポートされますが、カーソルを使って結果セットを更新する必要がない場合は、便利で柔軟性の高いスクロール可能カーソルを使用することをお勧めします。

カーソルをサポートするグローバル変数、コマンド、および関数の詳細については、『リファレンス・マニュアル：ビルディング・ブロック』および『リファレンス・マニュアル：コマンド』を参照してください。

カーソルを使用したローの選択

カーソルは、select 文に関連付けられ、次のもので構成されます。

- 「カーソル結果セット」− カーソルと関連のあるクエリの実行によって生じるローのセット (テーブル)。
- 「カーソル位置」− カーソル結果セット内の 1 つのローを示すポインタ。カーソルが読み込み専用として指定されていない場合、カーソルを指定する句が含まれている update 文または delete 文を使用してそのローを明示的に変更または削除できます。

センシビリティとスクロール対応

カーソルを宣言するときに、次の 2 つのキーワードを使用してセンシビリティを指定できます。

- insensitive
- semi_sensitive

カーソルを insensitive と宣言すると、そのカーソルではカーソルをオープンした時点の結果セットのみが表示され、基本となるテーブルでのデータの変更は表示されません。カーソルを semi_sensitive (デフォルト値) と宣言すると、カーソルをオープンした後で行われたベース・テーブルでの変更の一部が結果セットに表示されます。データの変更は semisensitive カーソルに表示される場合と表示されない場合があります。

スクロール対応を指定する次の 2 つのキーワードもあります。

- scroll
- no scroll

scroll を使用してカーソルを宣言した場合、結果のローを連続または非連続でフェッチできます。また、結果セットを繰り返しスキャンすることもできます。no scroll (デフォルト値) でカーソルを宣言すると、そのカーソルはスクロール不可です。結果セットは、一度に 1 ローだけ前方向へのみ表示されます。

どちらの属性も指定していない場合、デフォルト値は no scroll です。

カーソルは、select 文の結果セットの「ハンドル」と考えることができます。カーソルは、スクロール対応に応じて、連続または非連続でフェッチできます。

非スクロール可能カーソルは、前方向でのみフェッチできます。既にフェッチしたローに戻ることはできません。スクロール可能なカーソルは、後方または前方のいずれかの方向でフェッチできます。

スクロール可能カーソルを使用すると、カーソルがオープンしている状態のときには、fetch 文で first、last、absolute、next、prior、または relative オプションを指定することで、カーソル結果セットの任意の場所にカーソルの位置を設定できます。

結果セットの最後のローをフェッチするには、次のように入力します。

```
fetch last [from] <cursor_name>
```

すべてのスクロール可能カーソルは読み込み専用です。更新可能なすべてのカーソルはスクロール不可です。

カーソルのタイプ

カーソルには、次の 4 つのタイプがあります。

- 「クライアント・カーソル」－ Open Client の呼び出し (または Embedded SQL) によって宣言されます。Open Client は、Adaptive Server から返されたローを追跡し、アプリケーション用にバッファします。クライアント・カーソルの結果セットに対する更新と削除は、Open Client の呼び出しによってだけ実行できます。クライアント カーソルは、最も頻繁に使用する種類のカーソルです。
- 「実行カーソル」－ クライアント・カーソルのうち、ストアド・プロシージャによって定義されている結果セットを持つものは、実行カーソルと呼ばれます。ストアド・プロシージャではパラメータが使えます。パラメータ値は Open Client 呼び出しを通じて送信します。
- 「サーバ・カーソル」－ SQL で宣言されます。サーバ・カーソルをストアド・プロシージャで使用した場合、そのストアド・プロシージャを実行するクライアントはサーバ・カーソルを認識しません。クライアントに返される fetch の結果は、通常の select の結果と同じです。
- 「言語カーソル」－ Open Client を使わないで SQL で宣言されます。サーバ・カーソルと同様に、クライアントは言語カーソルを認識しません。また、結果は通常の select と同じフォーマットでクライアントに返されます。

カーソルの説明を簡単にするために、このマニュアルの例では言語カーソルとサーバ・カーソルだけを取り上げます。クライアント・カーソルや実行カーソルの例については、Open Client または Embedded SQL のマニュアルを参照してください。

カーソル・スコープ

カーソルの存在は、カーソルが使用されているコンテキストを参照している「スコープ」によって異なります。

- ユーザ・セッション内
- ストアド・プロシージャ内
- トリガ内

ユーザ・セッション内では、カーソルはユーザがセッションを終了するまでの間しか存在しません。ユーザがログオフすると、Adaptive Server はそのセッションで作成されたカーソルの割り付けを解除します。カーソルは、他のユーザが開始した追加のセッションでは存在しません。

カーソル名は、指定のテーブル内でユニークでなければなりません。Adaptive Server は、実行時にのみ特定のスコープ内で名前の重複を検出します。ストアド・プロシージャまたはトリガは、カーソルが 1 つだけ実行されている場合に、同一名の 2 つのカーソルを定義できます。たとえば、次のようなストアド・プロシージャでは、そのスコープ内で定義されているカーソルが names_crsr 1 つだけであるため、このストアド・プロシージャは正常に動作します。

```
create procedure proc2 @flag int
as
if @flag > 0
    declare names_crsr cursor
    for select au_fname from authors
else
    declare names_crsr cursor
    for select au_lname from authors
return
```

カーソルのスキャンとカーソル結果セット

Adaptive Server がカーソル結果セットの作成に使用する方法は、カーソルおよびカーソルの select 文のクエリ・プランによって決まります。ワークテーブルが必要ではない場合、Adaptive Server は、テーブルのインデックス・キーを使用してベース・テーブル内でカーソルを位置付けることによって、fetch を実行します。これは、fetch で指定したローの数が返される以外は select 文でも同様に実行されます。fetch の後、再びフェッチを実行するか、カーソルをクローズするまで、Adaptive Server はカーソルを次に有効なインデックス・キーに置きます。

すべてのスクロール可能カーソルおよび insensitive 非スクロール可能カーソルでは、カーソル結果セットを保持するワークテーブルが必要です。一部のクエリでも、カーソル結果セットを生成するためにワークテーブルが必要です。カーソルがワークテーブルを使用するかどうかを確認するには、set showplan, no exec on 文の出力をチェックします。

ワークテーブルが使用されるときは、カーソルの fetch 文で取り出されたローは実際のベース・テーブルのローの値を反映していないことがあります。たとえば、order by 句を指定して宣言されたカーソルは、通常、カーソル結果セットのローを並べるために、ワークテーブルを作成する必要があります。Adaptive Server は、ワークテーブルのローに対応するベース・テーブルのローをロックしないので、他のクライアントがこれらのベース・テーブルのローを更新できてしまいます。カーソル文からクライアントに返されたローは、ベース・テーブルのローと一致しなくなります。「カーソルとロック」(599 ページ) を参照してください。

一般に、デフォルトのカーソルと semi_sensitive カーソルのカーソル結果セットはどちらも、そのカーソルに fetch を実行してローが返されるときに生成されます。つまり、カーソルの select クエリは、通常の select クエリのように処理されます。「カーソル・スキャン」と呼ばれるこの処理によって所要時間が短縮され、アプリケーションが必要としないローを読み込む必要がなくなります。

Adaptive Server では、特に独立性レベル 0 の読み込みにおいて、カーソル・スキャンにテーブルのユニーク・インデックスを使用する必要があります。テーブルに IDENTITY カラムがあり、そのテーブル上にユニークでないインデックスを作成する場合は、identity in nonunique index データベース・オプションを使用して、テーブルのインデックス・キーに IDENTITY カラムを含め、テーブル上に作成されるすべてのインデックスがユニークになるようにします。このオプションは、論理的にはユニークでないインデックスを内部的にユニークにし、それらのインデックスを使用して更新可能なカーソルと独立性レベル 0 の読み込みを処理できるようにします。

インデックスのないテーブルが、現在のローの位置を移動させる別の処理によって更新されていなければ、それらのテーブルを参照するカーソルを使用できます。次に例を示します。

```
declare storinfo_crsr cursor
for select stor_id, stor_name, payterms
    from stores
    where state = "CA"
```

上記のカーソルを指定したテーブル stores にはインデックスがありません。Adaptive Server では、declare cursor 文で for update を指定していなければ、ユニーク・インデックスを持たないテーブルに対してカーソルを宣言できます。update でローの位置が変更されない場合、カーソル位置は次の fetch まで変わりません。

カーソルを更新可能にする方法

カーソルが更新可能な場合は、カーソルが返したローを更新または削除できます。カーソルが読み込み専用の場合、カーソルを更新または削除できません。デフォルトでは、Adaptive Server は、カーソルを読み込み専用と指定する前に、更新可能かどうかを判断します。

Adaptive Server 15.7 以降で select for update 構成パラメータを使用してカーソルを更新可能にする方法については、「select for update の使用」(45 ページ) を参照してください。

declare 文に read only キーワードまたは update キーワードを使用して、カーソルが読み込み専用かどうかを明示的に指定できます。カーソルを読み込み専用として指定すると、Adaptive Server は位置付け更新を正しく実行できます。更新するテーブルには、ユニーク・インデックスが必要です。これがないと、Adaptive Server は declare cursor 文を拒否します。

すべてのスクロール可能カーソルとすべての insensitive カーソルは読み込み専用です。

次の例では、pubs_crsr カーソルの更新可能な結果セットを定義します。

```
declare pubs_crsr cursor
for select pub_name, city, state
from publishers
for update of city, state
```

この例では、publishers テーブルのすべてのローが含まれていますが、city カラムと state カラムだけを更新用として明示的に定義しています。

カーソルを使用してローを更新または削除しない場合は、読み込み専用としてカーソルを宣言してください。read only か update を明示的に指定しない場合は、select 文に次の構造のいずれかが含まれていなければ、semi_sensitive 非スクロール可能カーソルは暗黙的に更新可能になります。

- distinct オプション
- group by 句

- 集合関数
- サブクエリ
- union 演算子
- at isolation read uncommitted 句

カーソルの select 文にこれらの構成体のいずれかが含まれている場合は、for update 句を指定できません。また、select 文の一部として order by 句を含むある種のカーソルを宣言すると、Adaptive Server はカーソルを読み込み専用として定義します。「カーソルのタイプ」(571 ページ) を参照してください。

更新できるカラムの判別

スクロール可能カーソルと insensitive 非スクロール可能カーソルは読み込み専用です。for update 句を使って *column_name_list* を指定しない場合は、クエリ内の指定されているカラムはすべて更新可能です。Adaptive Server は、ベース・テーブルをスキャンするときに、更新可能なカーソルのユニーク・インデックスを利用しようとします。カーソルに関しては Adaptive Server は、IDENTITY カラムを含むインデックスを、たとえそれが宣言されていなくても、ユニークであるとみなします。

Adaptive Server では、カーソルの select 文のカラム・リストには指定されていないが、select に指定されたテーブルの一部であるカラムを更新できます。ただし、for update で *column_name_list* を指定した場合は、そのリスト内のカラムしか更新できません。

次の例では、Adaptive Server は (pub_id が newpubs_crsr の定義に含まれていない場合でも) publishers の pub_id カラム上のユニーク・インデックスを使用します。

```
declare newpubs_crsr cursor
for select pub_name, city, state
from publishers
for update
```

for update 句が指定されていないと、Adaptive Server は任意のユニーク・インデックスを選択します。ただし、指定されたテーブル・カラムのユニーク・インデックスが存在しない場合、他のインデックスやテーブル・スキャンを使用することもできます。for update 句が指定されると、Adaptive Server はベース・テーブルをスキャンするための 1 つまたは複数のカラム用に定義されたユニーク・インデックスを使用します。ユニーク・インデックスが存在しない場合、Adaptive Server はエラーを返します。

ほとんどの場合、for update 句の *column_name_list* には、更新対象のカラムだけを指定してください。カーソルが for update 句で宣言されており、テーブルにユニーク・インデックスが 1 つしかない場合、そのカラムを for update *column_name_list* に指定できません。Adaptive Server は、カーソルのスキャン中にそのカラムを使用します。テーブルに複数のユニーク・インデックスがある場合は、インデックス・カラムを for update *column_name_list* に指定できます。すると、Adaptive Server は、*column_name_list* にない別のユニーク・インデックスを使用してカーソル・スキャンを実行できます。たとえば、次の declare cursor 文で使用されるテーブルにはカラム c3 にユニーク・インデックスが 1 つあるだけなので、そのカラムを for update のリストに指定する必要はありません。

```
declare mycursor cursor
for select c1, c2, 3
from mytable
for update of c1, c2
```

しかし、mytable に複数のユニーク・インデックスがある場合 (たとえばカラム c3 と c4) は、次のように、1 つのユニーク・インデックスを for update 句に指定してください。

```
declare mycursor cursor
for select c1, c2, 3
from mytable
for update of c1, c2, c3
```

c3 と c4 の両方を *column_name_list* に指定することはできません。一般に、Adaptive Server がカーソル・スキャンを実行するには、リストにない少なくとも 1 つのユニーク・インデックス・キーが必要です。

このようにして Adaptive Server がカーソル・スキャンでユニーク・インデックスを使用できるようにすると、「ハロウィーン問題」と呼ばれる更新の異常を防ぐことができます。ハロウィーン問題は、ベース・テーブルからローが返される順序を定義するカラム (つまり、ユニーク・インデックス・カラム) を、クライアントがカーソルによって更新するときに発生します。たとえば、Adaptive Server がインデックスを使用してベース・テーブルにアクセスし、インデックス・キーがクライアントによって更新される場合は、更新されたインデックス・ローは、インデックス内で移動し、カーソルによって再び読み込まれます。このローは、インデックス・キーがクライアントによって更新される場合と、更新されたインデックス・ローが結果セット内で下に移動する場合の 2 回、結果セットに表示されると考えられます。

ハロウィーン問題は、unique auto_identity index データベース・オプションを on に設定してテーブルを作成しても回避できます。『パフォーマンス &チューニング・シリーズ：クエリ処理と抽象プラン』の「第 8 章 カーソルの最適化」を参照してください。

Adaptive Serverのカーソルの処理方法

カーソルを使用してデータにアクセスすると、Adaptive Server は処理を次のオペレーションに分割します。

- **カーソルの宣言**　カーソルを宣言すると、Adaptive Server はカーソル構造体を作成します。ただし、カーソルがオープンするまでは、サーバはそのカーソル宣言からカーソルをコンパイルしません。

  「declare cursor の使用」(581 ページ) を参照してください。

  次のカーソル宣言 business_crsr (デフォルトの非スクロール可能カーソル) は、titles テーブルの中のすべてのビジネス書のタイトルと ID 番号を検出します。

  ```
  declare business_crsr cursor
  for select title, title_id
  from titles
  where type = "business"
  for update of price
  ```

  カーソルを宣言するときは、for update 句を使用して、位置付け更新を Adaptive Server が正しく実行できるようにします。この例では、カーソルを使って価格を変更できます。

  次の例では、スクロール可能カーソル `authors_scroll_crsr` を宣言します。このカーソルは、authors テーブルでカリフォルニア州の作家を検出します。

  ```
  declare authors_scroll_crsr scroll cursor
  for select au_fname, au_lname
  from authors
  where state = 'CA'
  ```

  スクロール可能カーソルは読み込み専用であるため、カーソル宣言で for update 句といっしょに使用することはできません。

- **カーソルのオープン**　ストアド・プロシージャ外で宣言されたカーソルをオープンすると、Adaptive Server はカーソルをコンパイルし、最適化されたクエリ・プランを生成します。次に、カーソルで定義されたローをスキャンするための事前操作を実行して、結果セットを返す準備ができます。

ストアド・プロシージャ内でカーソルを宣言すると、ストアド・プロシージャが最初に呼び出されたときに Adaptive Server でカーソルがコンパイルされます。また、Adaptive Server は最適化されたクエリ・プランを生成し、そのプランを後で使用できるように保管します。ストアド・プロシージャが再度呼び出されると、そのカーソルは既にコンパイル済み形式で存在しています。カーソルがオープンされると、Adaptive Server はスキャンを実行し結果セットを返すための事前操作を実行するだけで済みます。

注意 Transact-SQL 文はカーソルのオープン段階でコンパイルされるため、カーソルの宣言に関連するすべてのエラー・メッセージはカーソルのオープン段階で表示されます。

- **カーソルからのフェッチ** fetch コマンドは、コンパイルされたカーソルを実行して、カーソルに定義されている条件に一致する 1 つ以上のローを返します。デフォルトでは、fetch はローを 1 つだけ返します。

  非スクロール可能カーソルでは、最初の fetch は、カーソルの探索条件に一致する最初のローを返し、カーソルの現在の位置を保管します。最初の fetch からのカーソル位置を使用した 2 番目の fetch は、探索条件に一致する次のローを返し、その現在の位置を保管します。後続の各 fetch は、前の fetch のカーソル位置を使用して、次のカーソル・ローを見つけます。

  スクロール可能カーソルでは、fetch 文でフェッチ方向を指定することによって、任意のローを**フェッチ**し、結果セットの任意のローに現在のカーソル位置を設定できます。方向オプションは、first、last、next、prior、absolute、および relative です。スクロール可能カーソルの fetch は、前方および後方の両方向で実行でき、結果セットを繰り返しスキャンできます。

  fetch によって返されるローの数は、set cursor rows を使って変更できます。「1 つの fetch による複数のローの取得」(587 ページ) を参照してください。

  たとえば、次に示す fetch コマンドでは、titles テーブルの中の、ビジネス書の入っている最初のローのタイトルと ID 番号が表示されます。

```
fetch business_crsr

title                                  title_id
-----------------------------------    --------
The Busy Executive's Database Guide    BU1032

(1 row affected)
```

  2 回目に fetch business_crsr を実行すると、titles にある次のビジネス書のタイトルと ID 番号が表示されます。

次の例に示す、スクロール可能カーソルへの最初の fetch コマンドでは、authors テーブルの中の、カリフォルニア州の作家の入っている 10 番目のローが表示されます。

```
fetch absolute 10 authors_scroll_crsr
au_fname au_lname
--------------------
Akiko Yokomoto
```

方向オプション prior を指定した 2 番目の fetch では、10 番目のローの前のローが返されます。

```
fetch prior authors_scroll_crsr
au_fname au_lname
------------------
Chastity Locksley
```

- **ローの処理**　Adaptive Server は、現在のカーソル位置でカーソル結果セット (およびデータを提供した対応するベース・テーブル) 内のデータの更新または削除を行います。これらの操作は省略可能です。

  次に示す update 文は、ビジネス書の価格を 5 パーセント上げます。対象となるのは、business_crsr カーソルが現在指している本だけです。

  ```
  update titles
  set price = price * .05 + price
  where current of business_crsr
  ```

  カーソル・ローの更新によって、ロー内のデータが変更される場合もあれば、ローが削除される場合もあります。カーソルを使用してローを挿入することはできません。カーソルを使って実行された更新はすべて、カーソル結果セットに含まれる、対応するベース・テーブルに影響します。

- **カーソルのクローズ**　Adaptive Server は、カーソル結果セットをクローズして、残っているテンポラリ・テーブルをすべて削除し、カーソルの構造体に対して保持されているサーバ・リソースを解放します。ただし、カーソルを再びオープンできるように、カーソルのクエリ・プランは保持します。カーソルをクローズするには、close コマンドを使用します。次に例を示します。

  ```
  close business_crsr
  ```

  カーソルをクローズして再オープンすると、Adaptive Server はカーソル結果セットを再作成し、最初の有効なローの前にカーソルを置きます。これによって、カーソル結果セット全体にアクセスする必要はありません。カーソルは何回でもクローズでき、結果セット内全体を移動する必要はありません。

- **カーソルの割り付け解除** Adaptive Server は、メモリからクエリ・プランを削除し、カーソル構造体のトレースをすべて削除します。カーソルを割り付け解除するには、deallocate cursor コマンドを使用します。次に例を示します。

  ```
  deallocate cursor business_crsr
  ```

  Adaptive Server バージョン 15.0 以降では、キーワード cursor は、このコマンドのオプションです。

  使用する前に、カーソルを再度宣言する必要があります。

カーソル文のモニタ

Adaptive Server では、monCachedStatement テーブルを使用してカーソルをモニタします。monCachedStatement にある StmtType カラムは、文のキャッシュのクエリの種類を示しています。StmtType の値は次のとおりです。

- 1 − バッチ文
- 2 − カーソル文
- 3 − 動的文

注意 enable functionality group 設定パラメータを 1 に設定して、カーソル文をモニタする必要があります。

この例では、monCachedStatement (new_cursor の SSQLID を含む) の詳細を表示し、次に show_cached_text 関数を使用して次の new_cursor の SQL テキストを表示します。

```
InstanceID、SSQLID、Hashkey、UseCount、StmtType を
monCachedStatement から選択します。
InstanceID   SSQLID       Hashkey       UseCount      StmtType
 ----------  -----------  -----------   -----------  ----------
          0    329111220   1108036110             0           2
          0    345111277   1663781964             1           1

select show_cached_text(329111220)
--------------------------------------
select id from sysroles
```

『リファレンス・マニュアル：テーブル』の「第 3 章 モニタリング・テーブル」および『パフォーマンス&チューニング・シリーズ：モニタリング・テーブル』を参照してください。

declare cursor の使用

declare cursor 文は、カーソルの open 文よりも先に指定する必要があります。ストアド・プロシージャ内でカーソルを使用しているとき以外は、同じ Transact-SQL バッチ内で declare cursor 文を他の文と結合できません。

select_statement は、カーソル結果セットを定義するクエリです。『リファレンス・マニュアル：コマンド』を参照してください。一般に *select_statement* では、holdlock キーワードを含む、Transact-SQL の select 文の完全構文とセマンティクスを使用できます。しかし、これには compute、for browse、または into 句を指定することはできません。

カーソル・センシビリティ

insensitive または semi_sensitive のいずれかを使用して、カーソル・センシビリティを明示的に指定できます。

insensitive カーソルは、カーソルがオープンしているときに取得される結果セットのスナップショットです。カーソルをオープンすると、内部ワークテーブルが作成され、カーソル結果セットの値が完全に格納されます。

ベース・テーブルのロックは解放され、fetch を実行すると、ワークテーブルだけがフェッチ用に使われます。カーソルが宣言されたベース・テーブルでのデータの変更は、カーソルの結果セットに影響しません。カーソルは読み込み専用で、for update では使用できません。

semi_sensitive カーソルでは、ベース・テーブルでのデータ変更の一部がカーソルに表示されます。選択されたクエリ・プラン、およびデータ・ローが少なくとも 1 回フェッチされているかどうかが、ベース・テーブルのデータ変更の可視性に影響する場合があります。

semi_sensitive スクロール可能カーソルは、ワークテーブルを使用してスクロール用に結果セットを保持するという点で insensitive カーソルと似ています。semi_sensitive モードでは、カーソルのワークテーブルはカーソルがオープンされるときではなく、ローがフェッチされるときにマテリアライズされます。結果セットのメンバシップは、すべてのローが 1 回フェッチされ、スクロール用のワークテーブルにコピーされた後でのみ固定されます。

カーソル・センシビリティを指定しない場合、デフォルト値は semi_sensitive になります。

カーソルを semi_sensitive と宣言しても、そのカーソルのベース・テーブルにおけるデータ変更の可視性は、オプティマイザによって選択されているクエリ・プランによって異なります。

カーソルを semi_sensitive と宣言していても、任意の sort コマンドによってカーソルは強制的に insensitive になります。これは、sort を実行するためにテーブルのローが順番になっている必要があるためです。ただし、ローをフェッチする前にワークテーブルに値を格納できます。

たとえば、select 文で order by 句が指定され、order by カラムにインデックスがない場合は、カーソルを semi_sensitive と宣言しているかどうかに関係なく、カーソルがオープンする時点でワークテーブルには完全に値が格納されます。したがって、カーソルは insensitive になります。

一般に、まだフェッチされていないローではデータ変更が表示されるのに対し、既にフェッチされたローでは表示されません。

insensitive スクロール可能カーソルではなく semi_sensitive スクロール可能カーソルを使用する主な利点は、テーブル・ロックがローごとに適用されるため、結果セットの最初のローがすぐにユーザに返されることです。ローをフェッチして更新すると、そのローは fetch によってワークテーブルの一部になり、更新はベース・テーブルで行われます。結果セットのワークテーブルに完全に値が格納されるまで待つ必要はありません。

cursor_scrollability

scroll または no scroll のいずれかを使って、cursor_scrollability を指定できます。カーソルがスクロール可能な場合、任意つまり多数のローを交互にフェッチすることでカーソル結果セット内をスクロールできます。また、結果セットを繰り返しスキャンすることもできます。

すべてのスクロール可能カーソルは読み込み専用であり、カーソル宣言で for update といっしょに使用することはできません。

read_only オプション

read_only オプションは、カーソル結果セットを更新できないことを指定します。これに対して、for update オプションはカーソル結果セットが更新可能であることを指定します。for update の後には of *column_name_list* を指定できます。これには、更新可能として定義された *select_statement* のカラムのリストを指定します。

declare cursor の例

次の declare cursor 文では、カリフォルニア州以外に住むすべての作家を含む authors_crsr カーソルの結果セットを定義します。

```
declare authors_crsr cursor
for select au_id, au_lname, au_fname
from authors
where state != 'CA'
for update
```

次の例では、カリフォルニア州の書店を含む stores_scrollcrsr の insensitive スクロール可能結果セットを定義します。

```
declare storinfo_crsr insensitive scroll cursor
for select stor_id, stor_name, payterms
from stores
where state = "CA"
```

“C1” という insensitive の非スクロール可能カーソルを宣言するには、次のように入力します。

```
declare C1 insensitive cursor for
select fname from emp_tab
```

“C3” という insensitive のスクロール可能カーソルを宣言するには、次のように入力します。

```
declare C3 insensitive scroll cursor for
select fname from emp_tab
```

最初のローをフェッチするには、次のように入力します。

```
fetch last [from] <cursor_name>
```

また、結果セットから最初のローのカラムをフェッチすることもできます。*<fetch_target_list>* で指定した変数に最初のローのカラムを代入するには、次のように入力します。

```
fetch first from <cursor_name> into <fetch_target_list>
```

カーソルの現在の位置に関係なく、結果セットの 20 番目のローを直接フェッチできます。

```
fetch absolute 20 from <cursor_name> into <fetch_target_list>
```

カーソルのオープン

カーソルを宣言した後、ローの fetch、update、delete を行うには、カーソルをオープンする必要があります。カーソルをオープンすると、Adaptive Server は、カーソルを定義している select 文を評価してからカーソル結果セットの作成を開始し、処理できる状態にします。

open *cursor_name*

既にオープンしているカーソルや、declare cursor 文で定義されていないカーソルはオープンすることができません。クローズされたカーソルを再びオープンして、カーソル位置をカーソル結果セットの最初にリセットできます。

カーソルのタイプとクエリ・プランに応じて、カーソルをオープンしたときにワークテーブルが作成され、値が格納されます。

カーソルを使用したデータ・ローのフェッチ

fetch により、カーソル結果セットが完了し、1 つ以上のローがクライアントに戻ります。カーソルに定義されているクエリのタイプに応じて、Adaptive Server は直接テーブルをスキャンするか、そのクエリ・タイプとカーソル・タイプで生成されたワークテーブルをスキャンして、カーソル結果セットを作成します。

fetch コマンドを実行すると、カーソル結果セットの最初のローの前に非スクロール可能カーソルが置かれます。テーブルに有効なインデックスがある場合、Adaptive Server はカーソルを最初のインデックス・キーに置きます。

fetch 構文

first、next、prior、last、absolute、relative によって、スクロール可能カーソルの fetch 方向を指定します。キーワードを指定しない場合、デフォルト値は next です。『リファレンス・マニュアル：コマンド』を参照してください。

fetch absolute または fetch relative を使用する場合は、*fetch_offset* を指定してください。fetch_offset には、整数または位取りが 0 の符号付き真数値のリテラルを使用できます。または、位取りが 0 の、integer または numeric データ型の Transact-SQL ローカル変数を使用できます。カーソルを末尾のローの後または先頭のローの前に移動すると、データは返されず、エラーも生成されません。

fetch absolute を使用し、*fetch_offset* を 0 以上にすると、オフセットは結果セットの先頭ローの前の位置から計算されます。fetch absolute が 0 より小さければ、オフセットは結果セットの末尾ローの後の位置から計算されます。

fetch relative を使用する場合、*fetch_offset* n が 0 より大きければ、カーソルは現在の位置から *n* ロー後に配置されます。*fetch_offset n>0* のときは、カーソルは現在の位置から *abs(n)* ロー前に配置されます。

たとえば、スクロール可能カーソル *stores_scrollcrsr* を使用すると、任意のローをフェッチできます。この fetch によって、結果セットの 3 番目のローにカーソルが置かれます。

```
fetch absolute 3 stores_scrollcrsr
stor_id stor_name
---------------------------------------
7896 Fricative Bookshop
```

これに続く fetch prior 操作では、結果セットの 2 番目のローにカーソルが置かれます。

```
fetch prior stores_scrollcrsr
stor_id stor_name
---------------------------------------
```

```
7067 News & Brews
```

これに続く fetch relative -1 によって、結果セットの最初のローにカーソルが置かれます。

```
fetch relative -1 stores_scrollcrsr
stor_id stor_name
------------------------
7066     Barnum's
```

カーソル結果セットを生成した後、非スクロール可能カーソルの fetch 文で、Adaptive Server は結果セットの中で 1 行分カーソル位置を移動させます。Adaptive Server は結果セットからデータを取り出し、現在の位置を保管して、結果セットの終わりに達するまでフェッチを実行できるようにします。

次の例では、非スクロール可能カーソルを示します。authors_crsr カーソルを宣言してオープンしたら、その結果セットの最初のローを次のように fetch できます。

```
fetch authors_crsr

au_id        au_lname             au_fname
-----------  -------------------- ---------------
341-22-1782  Smith                Meander

(1 row affected)
```

引き続き fetch を実行するごとに、カーソル結果セットから次のローが取得されます。次に例を示します。

```
fetch authors_crsr

au_id        au_lname             au_fname
-----------  -------------------- ---------------
527-72-3246 Greene               Morningstar

(1 row affected)
```

ローをすべて fetch すると、カーソルは結果セットの最後のローを指します。もう一度 fetch を実行すると、Adaptive Server は、データがないことを示す *@@sqlstatus* または *@@fetch_status* グローバル変数 (「カーソル・ステータスのチェック」(586 ページ) で説明) を使って警告を返します。カーソル位置は変わりません。

非スクロール可能カーソルを使用している場合は、既にフェッチされたローでは fetch を実行できません。カーソルをクローズしてオープンし直し、カーソル結果セットをもう一度生成して、最初からフェッチを開始してください。

into 句の使用

into 句は、Adaptive Server が指定された変数にカラム・データを返すように指定します。*fetch_target_list* は、以前宣言した Transact-SQL パラメータかローカル変数で構成する必要があります。

たとえば、*@name*、*@city*、および *@state* 変数を宣言した後で、次のように pubs_crsr カーソルからローをフェッチできます。

```
fetch pubs_crsr into @name, @city, @state
```

また、結果セットから最初のローのカラムだけをフェッチすることもできます。フェッチ・カラムをリストに入れるには、次のように入力します。

```
fetch first from <cursor_name> into <fetch_target_list>
```

カーソル・ステータスのチェック

Adaptive Server は、各フェッチの後にステータス値を返します。グローバル変数 *@@sqlstatus*、*@@fetch_status*、または *@@cursor_rows* により値にアクセスできます。*@@fetch_status* と *@@cursor_rows* は、Adaptive Server バージョン 15.0 以降でのみサポートされます。

表 19-1 に、*@@sqlstatus* の値とその意味を示します。

表 19-1: @@sqlstatus 値

値	意味
0	fetch 文が正常に終了した。
1	fetch 文がエラーになった。
2	結果セットにこれ以上データがない。現在のカーソル位置が結果セットの最終ローにあり、クライアントがそのカーソルの fetch 文を実行すると、この警告が出される。

表 19-2 に、*@@fetch_status* の値とその意味を示します。

表 19-2: @@fetch_status 値

値	意味
0	fetch 操作が成功した。
-1	fetch 操作が失敗した。
-2	今後のために予約済み。

次の例では、現在オープンされている authors_crsr カーソルの *@@sqlstatus* を判定します。

```
select @@sqlstatus
---------
        0

(1 row affected)
```

次の例では、現在オープンされている authors_crsr カーソルの *@@fetch_status* を判定します。

```
select @@fetch_status

---------
```

```
           0

(1 row affected)
```

fetch 文だけが *@@sqlstatus* および *@@fetch_status* を設定できます。これ以外の SQL 文は、*@@sqlstatus* には作用しません。

@@cursor_rows は、最後にオープンされ、フェッチされたカーソル結果セットのローの数を示します。

表 19-3: @@cursor_rows 値

値	意味
-1	次のいずれかを示す。 • カーソルは動的。動的カーソルにはすべての変更が反映されるため、カーソルの条件を満たすローの数は常に変動する。条件を満たすローがすべて取得されたと断言できない。 • カーソルは semisensitive でスクロール可能だが、スクロール用ワークテーブルにまだ値が格納されていない。このため、結果セットの条件を満たすローの数は不明。
0	カーソルがオープンされていない、最後にオープンされたカーソルの条件を満たすローがない、または最後にオープンされたカーソルがクローズされているか割り付け解除されている。
n	最後にオープン、またはフェッチされたカーソル結果セットには、完全に値が格納されている。返される値 (n) は、カーソル結果セット内のローの総数。

1 つの *fetch* による複数のローの取得

set cursor rows コマンドを使用して、fetch が返すローの数を変更できます。ただし、このオプションは into 句を含む fetch には作用しません。

set cursor rows の構文は次のとおりです。

```
set cursor rows number for cursor_name
```

ここで *number* は、カーソルのローの数を指定します。*number* には、小数点のない数字リテラルまたは integer 型のローカル変数を指定できます。宣言する各カーソルのデフォルト設定は 1 です。カーソルがオープンしていてもクローズしていても、cursor rows オプションを set できます。

たとえば、authors_crsr カーソルにフェッチするローの数を、次のように変更できます。

```
set cursor rows 3 for authors_crsr
```

カーソルのローの数を設定すると、authors_crsr の各 fetch は 3 つのローを返します。

```
fetch authors_crsr

au_id        au_lname            au_fname
-----------  ------------------- ---------------
```

```
648-92-1872 Blotchet-Halls      Reginald
712-45-1867 del Castillo        Innes
722-51-5424 DeFrance            Michel

(3 rows affected)
```

カーソルは、フェッチされた最終ロー (例では、作家 Michel DeFrance) を指します。

一度に複数のローをフェッチできることは、クライアント・アプリケーションにとって特に便利な機能です。複数のローをフェッチする場合、Open Client または Embedded SQL は、クライアント・アプリケーションに送られたローをバッファに入れます。クライアントでは、依然としてローが 1 つずつアクセスされますが、各 fetch では、Adaptive Server への呼び出しが少なくなり、パフォーマンスが向上します。

フェッチされたローの数のチェック

@@rowcount グローバル変数を使うと、直前のフェッチまでにクライアントに返された結果セットのローの数をモニタできます。この変数は、ある時点までにカーソルが参照したローの総数を示します。

非スクロール可能カーソルでは、カーソル結果セットからすべてのローが読み込まれると、*@@rowcount* はその結果セットのローの総数を表します。ローの総数は、最後にフェッチされたカーソルの *@@cursor_rows* の最大値を表します。

次の例では、現在オープンされている authors_crsr カーソルの *@@rowcount* を判定します。

```
select @@rowcount
----------
         5

(1 row affected)
```

スクロール可能カーソルである場合、*@@rowcount* に最大値はありません。値は、fetch オペレーションのたびにフェッチの方向に関係なく増加します。

次の例は、authors_scrollcrsr、つまりスクロール可能 insensitive カーソルの *@@rowcount* 値を示しています。この結果セットでは、5 つのローがあるものとします。カーソルがオープンした後の *@@rowcount* の初期値は 0 です。結果セットのすべてのローはベース・テーブルからフェッチされ、ワークテーブルに保存されます。次の fetch 例では、すべてのローがワークテーブルからアクセスされます。

```
fetch last authors_scrollcrsr   @@rowcount = 1
fetch first authors_scrollcrsr  @@rowcount = 2
fetch next authors_scrollcrsr   @@rowcount = 3
fetch relative 2 authors_scrollcrsr @@rowcount = 4
```

```
fetch absolute 3 authors_scrollcrs  @@rowcount = 5
fetch absolute -2 authors_scrollcrsr  @@rowcount = 6
fetch first authors_scrollcrsr  @@rowcount = 7
fetch absolute 0 authors_scrollcrsr  @@rowcount =7
(nodatareturned)
fetch absolute 2 authors_scrollcrsr  @@rowcount = 8
```

カーソルを使用したローの更新と削除

カーソルが更新可能な場合、ローを更新または削除するには、update 文または delete 文を使用します。Adaptive Server は、カーソルを定義する select 文をチェックすることで、カーソルが更新可能かどうかを判断します。declare cursor 文の for update 句を指定して、カーソルを明示的に更新可能に定義することもできます。「カーソルを更新可能にする方法」(574 ページ) を参照してください。

カーソル結果セットのローの更新

update 文の where current of 句を使用して、現在のカーソル位置のローを更新できます。カーソル結果セットへの更新は、カーソルのローが取り出されたベース・テーブルのローにも影響します。

update...where current of の構文は、次のとおりです。

```
update [[database.]owner.]{table_name | view_name}
    set [[[database.]owner.]{table_name.| view_name.}]
        column_name1  =
            {expression1 | NULL | (select_statement)}
        [, column_name2  =
            {expression2 | NULL | (select_statement)}]...
    where current of cursor_name
```

set 句は、カーソルの結果セットのカラム名を指定し、新しい値を割り当てます。複数のカラム名と値のペアをリストする場合は、カンマで区切る必要があります。

table_name または *view_name* は、カーソルを定義する select 文の最初の from 句で指定されたテーブルかビューにしてください。この from 句が (ジョインを使用して) 複数のテーブルやビューを参照する場合は、実際に更新されるテーブルかビューだけを指定できます。

たとえば、pubs_crsr カーソルが現在指しているローは、次のようにして更新できます。

```
update publishers
set city = "Pasadena",
    state = "CA"
where current of pubs_crsr
```

更新後もカーソル位置は変わりません。別の SQL 文がカーソルの位置を移動させないかぎり、そのカーソル位置でローの更新を継続できます。

Adaptive Server では、カーソルの *select_statement* のカラム・リストに指定されていなくても、その文で指定されたテーブルの一部であるカラムを更新できます。ただし、for update で *column_name_list* を指定した場合は、そのリスト内のカラムしか更新できません。

カーソル結果セットのローの削除

delete 文の where current of 句を使用して、現在のカーソル位置のローを削除できます。カーソルの結果セットからローを削除すると、ローは基本となるデータベース・テーブルから削除されます。カーソルを使用すると、一度に 1 つのローしか削除できません。

delete...where current of の構文は、次のとおりです。

```
delete [from]
    [[database.]owner.]{table_name | view_name}
    where current of cursor_name
```

table_name または *view_name* は、カーソルを定義する select 文の最初の from 句で指定されたテーブルかビューにしてください。

たとえば、authors_crsr カーソルが現在指しているローは、次のように入力して削除できます。

```
delete from authors
where current of authors_crsr
```

from キーワードはオプションです。

注意 カーソルが更新可能であっても、ジョインを含む select 文で定義されたカーソルのローは削除できません。

カーソルからローを削除すると、Adaptive Server は、カーソルの結果セット内で、削除されたローの次のローの前にカーソルを置きます。ここでも fetch を使用して次のローにアクセスします。削除されたローがカーソル結果セットの最終ローである場合は、Adaptive Server は結果セットの最終ローの後にカーソルを置きます。

たとえば、この例で現在のロー (作家 Michel DeFrance) を削除してから、カーソル結果セット内の次の 3 人の作家をフェッチできます (cursor rows の設定は 3 のままであるとします)。

```
fetch authors_crsr

au_id        au_lname            au_fname
-----------  ------------------- ---------------
807-91-6654  Panteley            Sylvia
899-46-2035  Ringer              Anne
```

```
998-72-3567  Ringer              Albert

(3 rows affected)
```

カーソルを参照しないで、ベース・テーブルからローを削除することもできます。ベース・テーブルが変更されると、カーソル結果セットも変更されます。

カーソルのクローズと割り付け解除

カーソルの結果セットでの作業が終了したら、以下のコマンドでカーソルを close できます。

close *cursor_name*

カーソルをクローズしても、カーソルの定義は変更されません。再度そのカーソルをオープンすると、Adaptive Server は以前と同じクエリを使用して新しいカーソル結果セットを作成します。次に例を示します。

```
close authors_crsr
open authors_crsr
```

次に、カーソル結果セットの最初のローから開始して authors_crsr をフェッチできます。カーソルに関するすべての条件 (set cursor rows によって定義されたフェッチ済みローの数など) は、そのまま有効です。

カーソルを破棄するには、次を使用してそれを割り付け解除します。

deallocate cursor *cursor_name*

注意　Adaptive Server 15.0 以降では、cursor は オプションです。

カーソルを割り付け解除すると、カーソル名を含む、カーソルと関連のあるリソースがすべて解放されます。割り付け解除するまでは、カーソル名を再使用できません。オープン中のカーソルを割り付け解除すると、Adaptive Server は自動的にそのカーソルをクローズします。サーバへのクライアントの接続を終了した場合にも、オープン中のカーソルはクローズされ、割り付け解除されます。

前方にのみスクロール可能なカーソルの使用例

前方専用 (デフォルト) カーソル

ここでいうカーソル例では、次のクエリが使用されます。

```
select author = au_fname + " " + au_lname, au_id
from authors
```

クエリの結果は、次のとおりです。

```
author                      au_id
-------------------------   -----------
Johnson White               172-32-1176
Marjorie Green              213-46-8915
Cheryl Carson               238-95-7766
Michael O'Leary             267-41-2394
Dick Straight               274-80-9391
Meander Smith               341-22-1782
Abraham Bennet              409-56-7008
Ann Dull                    427-17-2319
Burt Gringlesby             472-27-2349
Chastity Locksley           486-29-1786
Morningstar Greene          527-72-3246
Reginald Blotchet Halls     648-92-1872
Akiko Yokomoto              672-71-3249
Innes del Castillo          712-45-1867
Michel DeFrance             722-51-5454
Dirk Stringer               724-08-9931
Stearns MacFeather          724-80-9391
Livia Karsen                756-30-7391
Sylvia Panteley             807-91-6654
Sheryl Hunter               846-92-7186
Heather McBadden            893-72-1158
Anne Ringer                 899-46-2035
Albert Ringer               998-72-3567

(23 rows affected)
```

次に、上記のクエリでカーソルを使用する方法を示します。

1 カーソルを宣言します。

次の declare cursor 文では、上に示した select 文を使用してカーソルを定義しています。

```
declare newauthors_crsr cursor for
select author = au_fname + " " + au_lname, au_id
from authors
for update
```

2　カーソルをオープンします。

```
open newauthors_crsr
```

3　カーソルを使用してローをフェッチします。

```
fetch newauthors_crsr

author                        au_id
-------------------------   -----------
Johnson White                172-32-1176

(1 row affected)
```

set cursor rows コマンドでローの数を指定して、一度に複数のローをフェッチできます。

```
set cursor rows 5 for newauthors_crsr
go
fetch newauthors_crsr

author                        au_id
-------------------------   -----------
Marjorie Green               213-46-8915
Cheryl Carson                238-95-7766
Michael O'Leary              267-41-2394
Dick Straight                274-80-9391
Meander Smith                341-22-1782

(5 rows affected)
```

fetch は、実行されるたびに次に続く 5 つのローを返します。

```
fetch newauthors_crsr

author                        au_id
-------------------------   -----------
Abraham Bennet               409-56-7008
Ann Dull                     427-17-2319
Burt Gringlesby              472-27-2349
Chastity Locksley            486-29-1786
Morningstar Greene           527-72-3246

(5 rows affected)
```

カーソルは、現在の fetch の最後のローである Morningstar Greene を指しています。

4　Greene の名前を変更するには、次のように入力します。

```
update authors
set au_fname = "Voilet"
where current of newauthors_crsr
```

カーソルは、次の fetch が実行されるまで、Ms. Greene のレコードにあります。

5 カーソルの使用が終了したら、次のようにしてカーソルをクローズできます。

```
close newauthors_crsr
```

カーソルを再び open すると、Adaptive Server はクエリを再実行して、カーソルを結果セットの最初のローの前に置きます。カーソルは、fetch ごとに 5 つのローを返すように設定されたままです。

6 カーソルを削除するには、次を使用します。

```
deallocate cursor newauthors_crsr
```

割り付け解除するまでは、カーソル名を再使用できません。

スクロール可能カーソル用のテーブルの例

この項の例は、スクロール可能カーソルによって実行します。表 19-4 のデータを生成するには、次のコマンドを実行します。

```
select emp_id, fname, lname
from emp_tab
where emp_id > 2002000
```

ベース・テーブル emp_tab は、*emp_id* フィールドにクラスタード・インデックスを持つデータロー・ロック・テーブルです。“Row position” は、結果セットの各ローの位置を表す値を持つ架空のカラムです。このテーブルの結果セットは、insensitive カーソルと semisensitive カーソルの両方について説明する次の項の例で使用します。

表 19-4: select 文の実行結果

ロー位置	emp_id	fname	lname
1	2002010	Mari	Cazalis
2	2002020	Sam	Clarac
3	2002030	Bill	Darby
4	2002040	Sam	Burke
5	2002050	Mary	Armand
6	2002060	Mickey	Phelan
7	2002070	Sam	Fife
8	2002080	Wanda	Wolfe
9	2002090	Nina	Howe
10	2002100	Sam	West

insensitive スクロール可能カーソル

insensitive カーソルを宣言してオープンすると、ワークテーブルが作成され、カーソル結果セットの値が完全に格納されます。ベース・テーブルのロックは解放され、ワークテーブルだけがフェッチ用に使われます。

カーソル CI を insensitive カーソルとして宣言するには、次のように入力します。

```
declare CI insensitive scroll cursor for
select emp_id, fname, lname
from emp_tb
where emp_id > 2002000

open CI
```

これで、スクロール用ワークテーブルには、表 19-4に示すデータが格納されます。名前を “Sam” から “Joe” に変更するには、次のように入力します。

```
.....
update emp_tab set fname = "Joe"
where fname = "Sam"
```

これで、ベース・テーブル emp_tab にある 4 つの “Sam” ローが消えて、4 つの “Joe” ローで置き換えられます。

```
fetch absolute 2 CI
```

カーソルはカーソル結果セットから 2 番目のローを読み込み、ロー 2 の “2002020, Sam, Clarac” を返します。カーソルは insensitive なので、更新された値はカーソルには見えません。また、返されるローの値 (“Sam”) は、表 19-4 のロー 2 の値と同じです。

この次のコマンドは条件を満たすさらに 1 つのロー (つまり、declare cursor のクエリ条件を満たすロー) をテーブル emp_tab に挿入しますが、ローのメンバシップはカーソル内で固定されているため、追加されるローはカーソル CI には見えません。たとえば、次のように入力します。

```
insert into emp_tab values (2002101, "Sophie", "Chen", .., ..., ...)
```

次の fetch コマンドは、カーソルをワークテーブルの最後までスクロールし、結果セットの最後のローを読み込んで、ローの値 “002100, Sam, West” を返します。この場合も、カーソルは insensitive なので、emp_tab に挿入された新しいローは、カーソル CI の結果セットに表示されません。

```
fetch last CI
```

semisensitive スクロール可能カーソル

semisensitive スクロール可能カーソルは、ワークテーブルを使用してスクロール用に結果セットを保持するという点で insensitive カーソルと似ています。ただし、semi_sensitive モードでは、カーソルのワークテーブルはカーソルがオープンされるときではなく、ローがフェッチされるときにマテリアライズされます。結果セットのメンバシップは、すべてのローが 1 回フェッチされた後でのみ固定します。

カーソルの CSI semisensitive かつスクロール可能を宣言するには、次のように入力します。

```
declare CSI semi_sensitive scroll cursor for
select emp_id, fname, lname
from emp_tab
where emp_id > 2002000

open CSI
```

結果セットの最初の各ローには、表 19-4 に示すデータが含まれます。カーソルは semisensitive なので、カーソルをオープンしたときにワークテーブルにコピーされるローはありません。最初のレコードをフェッチするには、次のように入力します。

```
fetch first CSI
```

カーソルは emp_tab から最初のローを読み込み、2002010, Mari, Cazalis を返します。このローはワークテーブルにコピーされます。次のローをフェッチするために、次のように入力します。

```
fetch next CSI
```

カーソルは emp_tab から 2 番目のローを読み込み、2002020, Sam, Clarac を返します。このローはワークテーブルにコピーされます。名前 “Sam” を “Joe” で置き換えるには、次のように入力します。

```
......
update emp_tab set fname = "Joe"
where fname = "Sam"
```

ベース・テーブル emp_tab にある 4 つの “Sam” ローが消え、代わって 4 つの “Joe” ローが表示されます。2 番目のローだけをフェッチするには、次のように入力します。

```
fetch absolute 2 CSI
```

カーソルは結果セットから 2 番目のローを読み込んで従業員 ID 2002020 を返しますが、返されるローの値は “Joe” ではなく “Sam” です。カーソルは semi-sensitive なので、このローはローの更新前にワークテーブルにコピーされています。返されるローは結果セットのスクロール用ワークテーブルから得られるため、update 文によるデータ変更はカーソルには見えません。

4 番目のローをフェッチするには、次のように入力します。

```
fetch absolute 4 CSI
```

カーソルは、結果セットから 4 番目のローを読み込みます。ロー 4 (2002040, Sam, Burke) は “Sam” が “Joe” に更新された後でフェッチされているため、返される従業員 ID 2002040 は Joe, Burke です。これで、3 番目と 4 番目のローがワークテーブルにコピーされました。

新しいローを追加するには、次のように入力します。

```
insert into emp_tab values (2002101, "Sophie", "Chen", .., ..., ...)
```

条件を満たすローがさらに 1 つ結果セットに追加されます。このローは、カーソルが semisensitive で、最後のローをまだフェッチしていないため、次の fetch 文では見えません。更新されたローをフェッチするために、次のように入力します。

```
fetch last CSI
```

fetch 文によって、結果セットの 2002101, Sophie, Chen が読み込まれます。

last オプションを指定した fetch を使用した後に、カーソル CSI の条件を満たすローをすべてワークテーブルにコピーしています。ベース・テーブル emp_tab のロックは解放され、カーソル CSI の結果セットは固定されます。この後の emp_tab のデータ変更は、CSI の結果セットには反映されません。

注意　ロック・スキーマとトランザクション独立性レベルもカーソルの可視性に影響します。上記の例は、デフォルトの独立性レベルであるレベル 1 に基づきます。

ストアド・プロシージャでのカーソルの使用

カーソルは、ストアド・プロシージャ内では特に便利です。カーソルを使用すると、数種類のクエリが必要なタスクを、1 つのクエリを使用するだけで実行できます。ただし、すべてのカーソル・オペレーションは、1 つのプロシージャ内で実行しなければなりません。ストアド・プロシージャは、プロシージャ内で宣言されなかったカーソルについては、open、fetch、close を行うことができません。ストアド・プロシージャのスコープ外では、カーソルは定義されていません。「カーソル・スコープ」(572 ページ) を参照してください。

たとえば、次のストアド・プロシージャ au_sales は、sales テーブル内の特定の作家の本の売れ行きがよかったかどうかを調べます。このストアド・プロシージャは、カーソルを使用して各ローを検査し、その情報を出力します。カーソルを使用しないと、同じ作業を実行するのに複数の select 文が必要になります。ストアド・プロシージャ外では、declare cursor を含むその他の文を同じバッチに組み込むことはできないのでしてください。

```
create procedure au_sales (@author_id id)
as

/* declare local variables used for fetch */
declare @title_id tid
declare @title varchar(80)
declare @ytd_sales int
declare @msg varchar(120)

/* declare the cursor to get each book written
    by given author */
declare author_sales cursor for
select ta.title_id, t.title, t.total_sales
from titleauthor ta, titles t
where ta.title_id = t.title_id
and ta.au_id = @author_id

open author_sales
fetch author_sales
        into @title_id, @title, @ytd_sales
if (@@sqlstatus = 2)
begin
    print "We do not sell books by this author."
    close author_sales
    return
end

/* if cursor result set is not empty, then process
    each row of information */
while (@@sqlstatus = 0)
begin
    if (@ytd_sales = NULL)
    begin
        select @msg = @title +
           " -- Had no sales this year."
        print @msg
    end
    else if (@ytd_sales < 500)
    begin
        select @msg = @title +
            " -- Had poor sales this year."
        print @msg
    end
```

```
            else if (@ytd_sales < 1000)
            begin
                select @msg = @title +
                    " -- Had mediocre sales this year."
                print @msg
            end
            else
            begin
                select @msg = @title +
                    " -- Had good sales this year."
                print @msg
            end

            fetch author_sales into @title_id, @title,
            @ytd_sales
        end
```

次に例を示します。

```
    au_sales "172-32-1176"
Prolonged Data Deprivation: Four Case Studies -- Had good sales this year.

(return status = 0)
```

「第 17 章 ストアド・プロシージャの使用」を参照してください。

カーソルとロック

サーバのバージョン 15.7 以降のカーソル・ロック

Adaptive Server バージョン 15.7 以降では、同じトランザクション内の後続の更新、および更新可能なカーソルのためにデータローロック・テーブルの排他ロックを行うための select for update がサポートされています。

select for update の詳細については、「select for update の使用」(45 ページ)および『リファレンス・マニュアル：コマンド』を参照してください。

select for update をカーソル・コンテキストで実行する場合、カーソル open と fetch 文はトランザクションのコンテキスト内でなければなりません。そうでない場合は、Adaptive Server が 15.7 より前の機能に戻ります。

サーバのバージョン 15.7 より前のカーソル・ロック

設定パラメータ select for update がサーバのバージョン 15.7 以降に設定されていない場合は、Adaptive Server は 15.7 より前のバージョンと同じカーソル・ロック・メカニズムを使用します。

「カーソルのロック」の方法は、他の Adaptive Server のロック方法と同じです。通常、データの読み込みを行う (select または readtext などの) 文では、コミットされていないトランザクションから変更データが読み込まれていないようにするためには、各データ・ページに共有ロックを使用します。データの更新を行う文では、変更する各ページに対して「排他ロック」を使用します。デッドロックを減少させ、同時実行性を向上させるために、Adaptive Server が更新ロックを排他ロックよりも優先させることがよくあります。これは、クライアントがページのデータを変更する場合に起こります。

更新可能なカーソルでは、Adaptive Server は、declare cursor の for update 句を指定して参照されるテーブルまたはビューをスキャンする場合に、デフォルトでは更新ロックを使用します。for update 句が含まれていて、リストが空の場合、*select_statement* の from 句で参照されるすべてのテーブルとビューは、デフォルトでは更新ロックがかかります。for update 句が含まれていない場合、参照されたテーブルとビューは共有ロックを受け取ります。「共有ロック」を使用する各テーブル名の後の from 句に shared キーワードを追加することによって、更新ロックの代わりに共有ロックを使用できます。

insensitive カーソルでは、ベース・テーブルのロックは、ワークテーブルに値が完全に格納された後で解放されます。semisensitive スクロール可能カーソルでは、ベース・テーブルのロックは、結果セットの最後のローが 1 回フェッチされた後で解放されます。

注意　Adaptive Server は、カーソル位置がデータ・ページから出ると、更新ロックを解放します。アプリケーションはクライアント・カーソルのローをバッファするので、対応するサーバ・カーソルはクライアント・カーソルとは異なるデータ・ローやデータ・ページを指していることがあります。この場合、最初のクライアントが for update オプションを使用したとしても、2 番目のクライアントは最初のクライアントの現在のカーソル位置を表すローを更新できます。

トランザクション内のカーソルが取得した排他ロックはすべて、そのトランザクションが終了するまで保持されます。holdlock キーワードや set isolation level 3 オプションを使用すると、これは共有ロックや更新ロックにも適用されます。ただし、close on endtran オプションを設定しないと、トランザクションが終了した後もカーソルはオープンしたままで、その現在のページ・ロックは有効なままになります。また、その後でローをフェッチするときに引き続きロックを取得することもできます。

『パフォーマンス＆チューニング・シリーズ：ロックと同時実行制御』を参照してください。

カーソル・ロック・オプション

更新可能なカーソルを定義した場合の、(select 文の) holdlock または shared オプションの指定には、次のような効果があります。

- どちらのオプションも指定しない場合、現在フェッチされているページのデータだけを読み込むことができます。他のユーザは、現在フェッチされているページを、カーソルやその他の方法で更新することはできません。他のユーザは、カーソルに使用されているテーブルにカーソルを宣言できますが、現在フェッチされているページに対して更新ロックを取得することはできません。
- shared オプションを指定した場合、現在フェッチされているページのデータだけを読み込むことができます。他のユーザは、現在フェッチされているページを、カーソルやその他の方法で更新することはできません。
- holdlock オプションを指定した場合、(現在のトランザクションで) フェッチされたすべてのページのデータを読み込むことができ、(トランザクション内でない場合は) 現在フェッチされているページのデータだけを読み込むことができます。現在フェッチされているページや、現在のトランザクションでフェッチされたページを、他のユーザがカーソルまたはその他の方法で更新することはできません。他のユーザは、カーソルに使用されているテーブルにカーソルを宣言できますが、現在フェッチされているページや現在のトランザクションでフェッチされたページに対して更新ロックを取得することはできません。
- 両方のオプションを指定した場合、(現在のトランザクションで) フェッチされたすべてのページのデータを読み込むことができ、(トランザクションでない場合は) 現在フェッチされているページのデータだけを読み込むことができます。他のユーザは、現在フェッチされているページを、カーソルやその他の方法で更新することはできません。

更新可能なカーソルの拡張トランザクション・サポート

15.7 より前のサーバ・バージョンのトランザクション・サポート

15.7 より前のバージョンでは、次のような場合に、for update 句で宣言され、トランザクション・コンテキスト内で開くカーソルが閉じます。

- カーソルが閉じる前にトランザクションを明示的にコミットする
- set close on endtran が設定される

詳細については、「カーソルおよび DML での select for update の使用」(45 ページ) を参照してください。

15.7 以降のサーバ・バージョンのトランザクション・サポート

バージョン 15.7 以降では、select for update が設定されている場合、トランザクションがコミットされた後で、Adaptive Server はオープン・カーソルに対する fetch 操作をサポートします。

カーソルを開いた場合、トランザクション・モードに基づいて別のロック・メカニズムが使用されます。

- 連鎖モード – 暗黙的にトランザクションが開始され、フェッチされているローの排他ロックが使用されます。fetch の後でトランザクションを commit する場合、次の fetch コマンドは新しいトランザクションを開始します。新しいトランザクションでフェッチされているローの排他的ロックが使用され続けます。
- 非連鎖モード – 排他的ロックが使用されるのは、カーソルを開く前に明示的な begin tran 文を実行した場合のみです。そうでない場合は、フェッチされているローに更新ロックが設定され、排他的ロックが次にフェッチされているローに設定されていないという警告が表示されます。

2 つの fetch コマンドの間またはカーソルを一度クローズしてから再度オープンするまでの間で commit を実行する場合、すべての排他的ロックが解放されます。次の fetch コマンドでは、トランザクション・モードに基づいてロックが設定されます。

- 連鎖モード – フェッチされているローに排他ロー・ロックが設定されます。また、特定の非最適化条件で更新ロー・ロックが設定されます。
- 非連鎖モード – フェッチされているローに更新ロー・ロックが設定されます。begin tran が fetch コマンドに先行する場合、排他ロー・ロックが設定されます。

更新ロー・ロックが設定されると、それらは次の場合にのみ解放されます。

- カーソルがクローズ – 独立性レベル 2 および 3。
- カーソルが次のローへ移動 – 独立性レベル 1。

カーソルに関する情報の取得

sp_cursorinfo を使用して、カーソルの名前、現在のステータス、結果カラムについての情報を取得できます。次の例では、authors_crsr についての情報が表示されます。

```
sp_cursorinfo 0, authors_crsr
Cursor name 'authors_crsr' is declared at nesting level '0'.
The cursor is declared as NON-SCROLLABLE cursor.
The cursor id is 851969.
The cursor has been successfully opened 1 times.
The cursor was compiled at isolation level 1.
```

```
The cursor is currently scanning at a nonzero isolation level.
The cursor is positioned on a row.
There have been 4 rows read, 0 rows updated and 0 rows deleted through this
cursor.
The cursor will remain open when a transaction is committed or rolled back.
The number of rows returned for each FETCH is 1.
The cursor is updatable.
This cursor is using 3432 bytes of memory.
There are 3 columns returned by this cursor.
The result columns are:
Name = 'au_id', Table = 'authors', Type = VARCHAR, Length = 11 (updatable)
Name = 'au_lname', Table = 'authors', Type = VARCHAR, Length = 40 (updatable)
Name = 'au_fname', Table = 'authors', Type = VARCHAR, Length = 20 (updatable)

Showplan output for the cursor:

QUERY PLAN FOR STATEMENT 1 (at line 1).
Optimized using Serial Mode

       STEP 1
           The type of query is DECLARE CURSOR.

           1 operator(s) under root

         |ROOT:EMIT Operator (VA = 1)
         |
         |  |SCAN Operator (VA = 0)
         |  |  FROM TABLE
         |  |  authors
         |  |  Using Clustered Index.
         |  |  Index : auidind
         |  |  Forward Scan.
         |  |  Positioning at start of table.
         |  |  Using I/O Size 2 Kbytes for data pages.
         |  |  With LRU Buffer Replacement Strategy for data pages.
```

次の例は、スクロール可能カーソルに関する情報を表示します。

```
sp_cursorinfo 0, authors_scrollcrsr

Cursor name 'authors_scrollcrsr' is declared at nesting level '0'.
The cursor is declared as SEMI_SENSITIVE SCROLLABLE cursor.
The cursor id is 786434.
The cursor has been successfully opened 1 times.
The cursor was compiled at isolation level 1.
The cursor is currently scanning at a nonzero isolation level.
The cursor is positioned on a row.
There have been 1 rows read, 0 rows updated and 0 rows deleted through this cursor.
The cursor will remain open when a transaction is committed or rolled back.
The number of rows returned for each FETCH is 1.
The cursor is read only.
This cursor is using 19892 bytes of memory.
```

```
There are 2 columns returned by this cursor.
The result columns are:
Name = 'au_fname', Table = 'authors', Type =VARCHAR, Length = 20 (not updatable)
Name = 'au_lname', Table = 'authors', Type = VARCHAR, Length = 40 (not updatable)
```

カーソルのステータスをチェックするもう 1 つの方法として、*@@sqlstatus*、*@@fetch_status*、*@@cursor_rows*、および *@@rowcount* グローバル変数を使用する方法があります。詳細については、「カーソル・ステータスのチェック」(586 ページ) と「フェッチされたローの数のチェック」(588 ページ) を参照してください。

『リファレンス・マニュアル：プロシージャ』を参照してください。

カーソルの代わりとしてのブラウズ・モードの使用

ブラウズ・モードでは、テーブル全体を検索し、ローの値を 1 行ずつ更新できます。ブラウズ・モードは、DB-Library およびホスト・プログラミング言語を使うフロントエンド・アプリケーションで使用されます。ブラウズ・モードは、Open Server アプリケーションや旧バージョンの Open Client ライブラリと互換性があります。しかし、新しい Client-Library アプリケーション (バージョン 10.0.x 以降) でのブラウズ・モードの使用はおすすめしません。カーソルを使用することにより、移植性が高く柔軟な方法で同じ機能を実現できるからです。また、ブラウズ・モードは Sybase 固有のものであるため、異機種環境には適しません。

ロー単位でテーブルの値を変更するには通常、データの更新にカーソルを使用します。Client-Library アプリケーションは、テーブルからのローのフェッチ中にそのテーブルを更新するなどのいくつかのブラウズ・モード機能を実装する Client-Library カーソルを使用できます。しかし、カーソルによって、選択しているテーブルでのロック競合が発生する場合があります。

ブラウズ・モードの詳細については、Open Client/Server マニュアルの dbqual 関数を参照してください。

テーブルのブラウズ

フロントエンド・アプリケーションでテーブルをブラウズするには、select 文の最後に for browse キーワードを追加します。次に例を示します。

Open Client アプリケーションの select *文の始まり*
. .
for browse
Completion of the Open Client application routine

テーブルは、そのローにタイムスタンプが記録されていれば、フロントエンド・アプリケーションでブラウズできます。

ブラウズ・モードの制限事項

for browse 句は、union 演算子を伴う文やカーソル宣言では使用できません。

for browse オプションを含む select 文では、キーワード holdlock は使用できません。

ブラウズ・モードでは、select 文の中のキーワード distinct は無視されます。

ブラウズ用の新規テーブルにタイムスタンプを設定する

ブラウズ用に新しいテーブルを作成するときは、timestamp というカラムをテーブル定義に指定してください。このカラムには、自動的に timestamp データ型が割り当てられます。次に例を示します。

```
create table newtable(col1 int, timestamp,
    col3 char(7))
```

ローを挿入または更新すると、Adaptive Server は timestamp カラムにユニークな varbinary 値を、そのローに自動的に割り当てます。

既存のテーブルにタイムスタンプを設定する

ブラウズ用に既存のテーブルを準備する場合は、alter table を使って timestamp というカラムを追加します。次に例を示します。

```
alter table oldtable add timestamp
```

既存のローにはそれぞれ、null 値を持つ timestamp カラムが追加されます。タイムスタンプを生成するには、新しいカラム値を指定しないで各ローを更新します。

次に例を示します。

```
update oldtable
set col1 = col1
```

timestamp の値の比較

フロントエンド・アプリケーションでブラウズ・モードを使っている場合、タイムスタンプを比較するには、tsequal システム関数を使用します。たとえば、次の文は、ブラウズされた publishers 内のローを更新します。また、ブラウズされたローの timestamp カラムが、保管されたローの 16 進数のタイムスタンプと比較されます。2 つのタイムスタンプが同じでない場合はエラー・メッセージが返され、ローの更新は実行されません。

```
update publishers
set city = "Springfield"
where pub_id = "0736"
and tsequal(timestamp,0x0001000000002ea8)
```

where 句で tsequal 関数を探索引数として使用しないでください。tsequal を使用すると、where 句の残りの部分は 1 つのローにユニークに対応します。tsequal 関数は、insert と update 文でだけ使用してください。timestamp カラムを検索句として使用する場合は、通常の varbinary カラムのように、*timestamp1 = timestamp2* として比較してください。

第 20 章

トリガ：参照整合性

トリガを使用して、さまざまな動作を自動的に実行できます。たとえば、関連するテーブルを通しての変更のカスケード、カラム制限の実行、データ修正結果の比較、データベース間での参照整合性の管理などの動作が挙げられます。

トリガの動作

トリガは自動的に起動されます。トリガは、ユーザによるデータ入力であれ、アプリケーションによる動作であれ、何によってデータが修正されたかにかかわらず、自動的に起動されます。トリガは、1 つまたは複数のデータ修正オペレーション (update、insert、delete) に固有のものです。トリガは、1 つの SQL 文につき 1 回実行されます。

たとえば、ユーザが publishers テーブルから出版社を削除できないようにするには、次を使用できます。

```
create trigger del_pub
on publishers
for delete
as
begin
    rollback transaction
    print "You cannot delete any publishers!"
end
```

次にユーザが publishers テーブルからローを削除しようとすると、del_pub トリガが削除を取り消して、トランザクションをロールバックし、メッセージを出力します。

データ修正文が完了し、Adaptive Server がデータ型、ルール、または整合性の制約の違反を検査した後でなければ、トリガは「起動」しません。トリガと、トリガを起動する文は、単一のトランザクションとみなされ、そのトリガ内からロールバックできます。Adaptive Server が重大なエラーを検出すると、トランザクション全体がロールバックされます。

トリガの使用目的

- データベース内の関連テーブルを使用した変更のカスケード。たとえば、titles テーブル内にある title_id カラムを削除するトリガは、ほかのテーブルの対応するローを削除できます。このとき、titleauthor および roysched のローを見つけるためのユニーク・キーとして、title_id カラムを使用します。
- 参照整合性に違反する変更は、データ修正トランザクションを取り消すことによって、却下、つまりロールバックされます。このようなトリガは、プライマリ・キーと一致しない外部キーを挿入すると起動します。たとえば、新しい titleauthor.title_id 値が titles.title_id にある値と一致しない場合は、この挿入をロールバックする挿入トリガを titleauthor に作成できます。
- ルールでの定義よりも非常に複雑な制限を実行する。ルールとは異なり、トリガを使用するとカラムまたはデータベース・オブジェクトを参照できます。たとえば、トリガにより、本の値段を前払い金の 1% より多く値上げする更新をロールバックできます。
- “what if” 分析の実行たとえばトリガは、テーブルのデータの変更前と変更後の状態を比較し、その比較結果に基づいて動作を実行できます。

トリガの使用と整合性制約の比較

トリガを使用する代わりに、create table 文の参照整合性の制約を使用して、データベース内にあるテーブル全体の参照整合性を実行できます。ただし、参照整合性の制約の場合、次のことはできません。

- データベース内の関連テーブルを使用した変更のカスケード
- 他のカラムやデータベース・オブジェクトの参照による複雑な制約の実行
- “what if” 分析の実行

また、参照整合性の制約は、データの整合性を実行した結果として、現在のトランザクションをロールバックしません。トリガを使用して、参照整合性をどのように扱うかに応じてトランザクションをロールバックまたは継続することができます。「第 23 章 トランザクション：データの一貫性およびリカバリ」

上記のタスクのいずれかがアプリケーションに必要な場合は、トリガを使用してください。それ以外の場合は、参照整合性の制約を使用して、データの整合性を保持します。Adaptive Server は、制約に違反するデータ変更文もトリガを起動することがないように、トリガの前に参照整合性の制約を確認します。参照整合性制約の詳細については、「第 8 章 データベースおよびテーブルの作成」を参照してください。

トリガの作成

トリガは、データベース・オブジェクトです。トリガを作成するには、トリガを「起動」つまりアクティブにするテーブルおよびデータ修正コマンドを指定します。次に、トリガに実行させる任意の動作を指定します。

たとえば、titles テーブル内のデータを挿入、削除、更新するたびに、トリガによってメッセージが表示されるようにするには、次のようにします。

```
create trigger t1
on titles
for insert, update, delete
as
print "Now modify the titleauthor table the same way."
```

注意　deltitle トリガを除いて、この章で説明するトリガは、Adaptive Server に備えられている pubs2 データベースには含まれていません。この章に出てくる例を使用するには、create trigger 文を入力して、トリガの各例を作成してください。テーブルまたはカラムに対して、同じオペレーション (insert、update、または delete) の新しいトリガを実行するたびに、前回の操作が警告なしで上書きされます。

create trigger 構文

create 句は、トリガを作成して名前を付けます。トリガ名は、識別子の規則に準拠する必要があります。

on 句では、トリガをアクティブにするテーブルの名前を指定します。このテーブルは「トリガ・テーブル」と呼ばれることがあります。

トリガは他のデータベースのオブジェクトを参照できますが、作成されるのは現在のデータベース内です。トリガ名を修飾する所有者名は、テーブルの所有者名と同じにします。テーブルの所有者のみがテーブルにトリガを作成できます。create trigger 句または on 句のテーブル名にテーブル所有者を指定した場合は、ほかの句でも必ず指定してください。

for 句は、トリガ・テーブルのどのデータ修正コマンドがトリガを起動するかを指定します。前の例では、titles テーブルの insert、update、または delete によって、メッセージが出力されます。

SQL 文は、「トリガ条件」と「トリガ動作」を指定します。トリガ条件は、insert、delete、または update によってトリガ動作が実行されるかどうかを決める、追加の基準を指定します。1 つの if 句に複数のトリガ動作を指定する場合は、begin と end でグループ化します。

if update 句は、指定したカラムへの挿入または更新に対するテストを行います。更新の場合、その更新によってカラムの値を変更しなくても、update 文の set 句にカラム名が含まれているとき、if update 句は真 (true) とみなします。if update 句は、delete とともに使用しないでください。複数のカラムを指定でき、また、1 つの create trigger 文に対して複数の if update 句を使用できます。on 句でテーブル名を指定するため、if update 句で指定するカラムの前にはテーブル名を使用しないでください。『リファレンス・マニュアル：コマンド』を参照してください。

トリガで使用できない SQL 文

これらの文はトリガで使用できません。

- すべての create コマンド (create database、create table、create index、create procedure、 create default、 create rule、 create trigger、create view)
- すべての drop コマンド
- alter table と alter database
- truncate table
- grant と revoke
- update statistics
- reconfigure
- load database と load transaction
- disk init、disk mirror、disk refit、disk reinit、disk remirror、disk unmirror
- select into

トリガの使用による参照整合性の維持

トリガを使用して、参照整合性を維持できます。参照整合性とは、指定したデータのユニークな識別子のようにデータベース内の重要なデータを、データベースが変更されても正しいデータとして使用できるようにすることです。参照整合性は、プライマリ・キーおよび外部キーを使用して調整されます。

「プライマリ・キー」は、ローをユニークに識別する値を持つ 1 つのカラムまたは複数のカラムの組み合わせです。プライマリ・キーの値には、null を設定できず、ユニークなインデックスが含まれている必要があります。プライマリ・キーを持つテーブルは、他のテーブルの外部キーとジョインできます。プライマリ・キー・テーブルは「マスタ・ディテール関係」の「マスタ・テーブル」とみなすことができます。データベースの中には、通常このようなマスタ・ディテール・グループがたくさんあります。

sp_primarykey を使用して sp_helpjoinsで使用できるプライマリ・キーにマークを付け、syskeys テーブルに追加できます。

たとえば、title_id カラムは titles テーブルのプライマリ・キーです。このカラムは titles テーブル内の本をユニークに識別し、titleauthor、salesdetail、roysched の title_id とジョインされます。titles テーブルは titleauthor、salesdetail、roysched のマスタ・テーブルです。

「外部キー」は、プライマリ・キーと一致する値を持つカラム、または複数のカラムの組み合わせです。外部キーはユニークである必要はありません。1 つのプライマリ・キーに、複数の外部キーが対応することがあります。外部キー値は、プライマリ・キー値のコピーでなければなりません。つまり、プライマリ・キーにない値を持つ外部キーは存在しません。プライマリ・キー値と異なる場合には、外部キーの値は null になります。複合外部キーの一部が null の場合は、その外部キー全体の値は null になります。外部キーを持つテーブルは、マスタ・テーブルの「ディテール」テーブルまたは「従属」テーブルと呼ばれます。

sp_foreignkey を使用して、データベース内の外部キーにマークを付けることができます。これによって、syskeys テーブルを参照する sp_helpjoins プロシージャおよびほかのプロシージャを使用するように、データベースの外部キーにフラグを立てます。titleauthor、salesdetail、roysched内の title_id カラムは外部キーです。つまり、これらのテーブルはディテール・テーブルです。ほとんどの場合は、参照制約を使用して、テーブル間の参照整合性を維持できます (参照制約とは、特定のカラムに挿入されるデータが一致する値を別のテーブルに必ず持つようにする制約のこと)。これは、1 つのテーブルに許可される最大参照数が、200 に制限されているためです。テーブルがこの制限を超えるか、または特別な参照整合性が必要な場合は、参照整合性トリガを使用してください。

参照整合性トリガは、外部キーの値をプライマリ・キーの値と同期した状態に保ちます。データ修正がキー・カラムに影響する場合には、トリガは、「トリガ・テスト・テーブル」と呼ばれるテンポラリのワークテーブルを使用して、新しいカラムの値と関連するキーとを比較します。トリガを記述するときは、トリガ・テスト・テーブルに一時的に保管されるデータに基づいて比較を行います。

トリガ・テスト・テーブルを使用したデータ修正のテスト

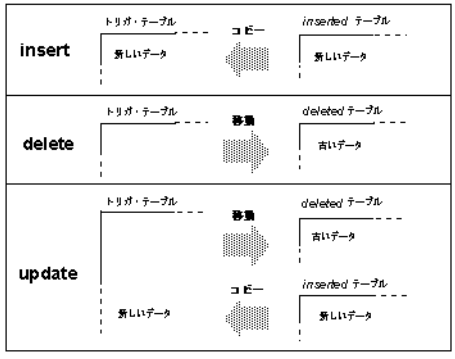
Adaptive Server では、トリガ文に 2 つの特殊なテーブル (deleted テーブルおよび inserted テーブル) を使用します。これらは、トリガ・テストに使用されるテンポラリ・テーブルです。トリガを作成するとき、これらのテーブルを使用して、データ修正の影響をテストしたり、トリガ動作の条件を設定できます。トリガ・テスト・テーブル内のデータは直接変更できませんが、select 文にトリガ・テスト・テーブルを指定して、insert、update、または delete の影響を調べることができます。

- deleted テーブルには、delete および update 文の実行中に影響を受けたローのコピーが保管されます。delete または update 文の実行中、ローはトリガ・テーブルから削除され、deleted テーブルに転送されます。通常、deleted テーブルとトリガ・テーブルに共通なローは存在しません。
- inserted テーブルには、insert および update 文の実行中に影響を受けたローのコピーが保管されます。insert または update の実行中、新しいローは、inserted テーブルとトリガ・テーブルの両方に同時に追加されます。inserted テーブルのローは、トリガ・テーブルの新しいローのコピーです。inserted テーブルを使用して titles テーブルの title_id カラムの変更をテストするためのトリガの一部を次に示します。

```
if (select count(*)
    from titles, inserted
    where titles.title_id = inserted.title_id) !=
    @@rowcount
```

注意 inserted テーブルも deleted テーブルも、トランザクション・ログにはビューとして表示されますが、syslogs ではどちらも偽のテーブルです。

update とは、実際は削除して挿入を行うことです。最初に、既存のローが deleted テーブルにコピーされ、次に inserted テーブルとトリガ・テーブルに新しいローがコピーされます。次の図は、insert、delete、および update 時のトリガ・テスト・テーブルの状況を示しています。



トリガ条件を設定するときは、データ修正に適したトリガ・テスト・テーブルを使用してください。insert のテスト中に deleted テーブルを参照したり、また、delete のテスト中に inserted テーブルを参照したりしてもエラーにはなりません。ただし、これらのトリガ・テスト・テーブルはローを含みません。

注意 1 つのトリガは、クエリ 1 つにつき一度だけ起動されます。データ修正による影響を受けたローの数によってトリガの動作を変える場合は、複数ローのデータ修正について *@@rowcount* の検査など、テストを使用し、適切な動作を実行してください。

以降のトリガの例では、必要に応じて複数ローのデータを修正します。最新のデータ修正オペレーションによる「影響を受けたローの数」を格納する *@@rowcount* 変数は、複数ローに対する insert、delete、または update のテストを行います。トリガ内で *@@rowcount* のテストより前に他の select 文がある場合、後でこの変数を検査するには、ローカル変数に値を格納してください。値を返さないすべての Transact-SQL 文ではすべて、*@@rowcount* が 0 にリセットされます。

挿入トリガの例

新しい外部キーのローを挿入する場合は、外部キーがプライマリ・キーと一致していることを確認してください。挿入トリガは、挿入されたロー (inserted テーブルで使用) と、プライマリ・キー・テーブルのロー間のジョインが確認されてから、プライマリ・キー・テーブルのキーと一致しない外部キーの挿入をロールバックします。

次のトリガは、inserted テーブルの title_id 値を titles テーブルのそれらの値と比較します。ここでは、外部キーに null 値以外のエントリが 1 つ入力されているものと仮定します。ジョインが失敗すると、トランザクションはロールバックされます。

```
create trigger forinsertrig1
on salesdetail
for insert
as
if (select count(*)
    from titles, inserted
    where titles.title_id = inserted.title_id) !=
    @@rowcount
/* Cancel the insert and print a message.*/
begin
    rollback transaction
    print "No, the title_id does not exist in
    titles."
end
/* Otherwise, allow it.*/
else
```

```
        print "Added!All title_id's exist in titles."
```

@@rowcount は、salesdetail テーブルに追加されたローの数です。この数は、inserted テーブルに追加されたローの数でもあります。トリガは、titles と inserted をジョインして、salesdetail に追加した title_id がすべて titles テーブルに存在するかどうかを確かめます。select count(*) クエリで判別したジョイン済みのローの数が *@@rowcount* と異なる場合は、1 つまたは複数の挿入が正しくないので、トランザクションは取り消されます。

挿入トリガは、挿入をロールバックする場合と受け入れる場合とでは、それぞれ異なるメッセージを表示します。最初の条件をテストするには、次の insert 文を実行します。

```
insert salesdetail
values ("7066", "234517", "TC9999", 70, 45)
```

2 番目の条件をテストするには、次を入力します。

```
insert salesdetail
values ("7896", "234518", "TC3218", 75, 80)
```

削除トリガの例

プライマリ・キー・ローを削除する場合は、従属テーブルにある、削除するプライマリ・キー・ローに対応する外部キーを削除してください。これによって、マスタのローが削除されるときにディテール・ローが取り除かれるため、参照整合性が保持されます。従属テーブル内の対応するローを削除しないと、取得および識別ができないディテール・ローがあるデータベースになってしまいます。削除を適切に行うには、カスケード delete を実行するトリガを使用します。

カスケード型削除の例

titles テーブルで delete 文が実行されると、1 つまたは複数のローが titles テーブルから削除されて deleted テーブルに追加されます。titles テーブルから削除されて deleted テーブルに保管されている title_id に一致する title_id を持つローが、titleauthor、salesdetail、roysched などの従属テーブルにあるかどうかを、トリガを使用して確認できます。トリガによってこのようなローが従属テーブルから検出された場合は、これらのローは削除されます。

```
create trigger delcascadetrig
on titles
for delete
as
delete titleauthor
from titleauthor, deleted
where titleauthor.title_id = deleted.title_id
/* Remove titleauthor rows that match deleted
** (titles) rows.*/
delete salesdetail
```

```
from salesdetail, deleted
where salesdetail.title_id = deleted.title_id
/* Remove salesdetail rows that match deleted
** (titles) rows.*/
delete roysched
from roysched, deleted
where roysched.title_id = deleted.title_id
/* Remove roysched rows that match deleted
** (titles) rows.*/
```

制限付き削除の例

実際においては、ディテール・ローをいくつか保持したい場合があるかもしれません。それは、履歴を残すことを目的としているか (絶版になったタイトルに関して、出版されていた間に何冊の売り上げがあったかどうか確認するため)、またはディテール・ローのトランザクションがまだ終了していない場合です。細かく記述されているトリガは、これらの要因を考慮に入れて書かれています。

プライマリ・キーの削除の防止

pubs2 の deltitle トリガは、salesdetail テーブルにプライマリ・キー用のディテール・ローがある場合に、プライマリ・キーの削除を防止するように書かれています。このトリガによって、salesdetail テーブルからローを取得する機能が保持されます。

```
create trigger deltitle
on titles
for delete
as
if (select count(*)
    from deleted, salesdetail
    where salesdetail.title_id =
    deleted.title_id) > 0
begin
    rollback transaction
    print "You cannot delete a title with sales."
end
```

このトリガでは、titles から削除された 1 つまたは複数のローが、salesdetail テーブルとジョインされることによってテストされます。ジョインが検出された場合は、トランザクションが取り消されます。

同様に、次の制限付き削除は、プライマリ・テーブル titles の従属する子が titleauthor にある場合は削除を防止します。deleted と titleauthor からのローを数える代わりに、title_id が削除されたかどうかをチェックします。この方法では、テーブル全体を調べてローをすべて数えるのではなく、特定のローの存在だけを調べるので、パフォーマンスの面でより効率的です。

発生したエラーの記録

次の例では、エラー・メッセージ 35003 に対して raiserror コマンドを使用します。raiserror は、エラーの発生を記録するためのシステム・フラグを設定します。この例を実行する場合は、実行の前にエラー・メッセージ 35003 を sysusermessages システム・テーブルに追加してください。

```
sp_addmessage 35003, "restrict_dtrig - delete failed: row
exists in titleauthor for this title_id."
```

トリガは次のとおりです。

```
create trigger restrict_dtrig
on titles
for delete as
if exists (select * from titleauthor, deleted where
          titleauthor.title_id = deleted.title_id)
   begin
           rollback transaction
           raiserror 35003
           return
   end
```

このトリガをテストするには、次の delete 文を実行します。

```
delete titles
where title_id = "PS2091"
```

更新トリガの例

次の例では、プライマリ・テーブル titles から従属テーブル titleauthor と roysched へ、更新をカスケードします。

```
create trigger cascade_utrig
on titles
for update as
if update (title_id)
begin
     update titleauthor
           set title_id = inserted.title_id
           from titleauthor, deleted, inserted
           where deleted.title_id = titleauthor.title_id
     update roysched
           set title_id = inserted.title_id
           from roysched, deleted, inserted
           where deleted.title_id = roysched.title_id
     update salesdetail
           set title_id = inserted.title_id
           from salesdetail, deleted, inserted
           where deleted.title_id = salesdetail.title_id
end
```

『Secrets of Silicon Valley』という本が、popular_comp から心理学の本に分類し直されたと想定して、このトリガをテストします。次のクエリは、title_id の PC8888 を、titleauthor、roysched、titles の PS8888 に更新します。

```
update titles
set title_id = "PS8888"
where title_id = "PC8888"
```

制限付き更新トリガ

プライマリ・キーは、そのテーブルのローと他のテーブルの外部キーのローをユニークに識別します。通常、プライマリ・キーの更新は許可しません。また、プライマリ・キーの更新は慎重に行う必要があります。更新する場合、指定された条件に合わないときは、更新をロールバックして参照整合性を保護してください。

たとえば、そのカラム上のパーミッションをすべて無効にするなど、プライマリ・キーの変更は行わないことをおすすめします。しかし、更新の禁止を特定の状況に限定する場合は、トリガを使用してください。

日付関数を使用した制限つき更新トリガ

次のトリガは、週末に titles.title_id が更新されないようにします。つまり stopupdatetrig 内の if update 句によって、titles.title_id という特定のカラムに対して変更を行わないように指定できます。このカラム内のデータを変更すると、トリガが起動します。他のカラムのデータを更新しても、トリガは起動しません。このトリガによって条件に違反する更新が検出されると、更新は取り消されメッセージが表示されます。これをテストするには、“Saturday” または “Sunday” の代わりに別の曜日を指定してください。

```
create trigger stopupdatetrig
on titles
for update
as
/* If an attempt is made to change titles.title_id
** on Saturday or Sunday, cancel the update.*/
if update (title_id)
      and datename(dw, getdate())
      in ("Saturday", "Sunday")
begin
    rollback transaction
    print "We do not allow changes to"
    print "primary keys on the weekend."
end
```

複数の動作を含む制限付き更新動作

if update を使用して、複数のカラムに複数のトリガ動作を指定できます。次の例は、stopupdatetrig を変更して、更新のためのトリガ動作を titles.price または titles.advance に追加します。この例は、プライマリ・キーの更新が週末に行われないようにします。また、タイトルの全収入が前払い金を超えないかぎり、そのタイトルの価格や前払い金の更新も行われないようにします。変更されたトリガは、再度作成するときに古いトリガを置き換えるので、同じトリガ名を使用できます。

```
create trigger stopupdatetrig
on titles
for update
as
if update (title_id)
  and datename(dw, getdate())
  in ("Saturday", "Sunday")
begin
```

```
        rollback transaction
        print "We do not allow changes to"
        print "primary keys on the weekend!"
    end
    if update (price) or update (advance)
    if exists (select * from inserted
        where (inserted.price * inserted.total_sales)
        < inserted.advance)
        begin
          rollback transaction
          print "We do not allow changes to price or"
          print "advance for a title until its total"
          print "revenue exceeds its latest advance."
        end
```

次のいずれかの条件が true の場合に update を実行できないようにするトリガを、titles 上に作成する例を示します。

- ユーザが、titles 内のプライマリ・キー title_id の値を変更しようとした。
- 従属キー pub_id が、publishers 内で見つからない。
- ターゲット・カラムが存在しないか、または値が null である。

この例を実行する前に、次のエラー・メッセージが sysusermessages に存在することを確認してください。

```
sp_addmessage 35004, "titles_utrg - Update Failed: update of primary keys %1! is not
allowed."
sp_addmessage 35005, "titles_utrg - Update Failed: %1! not found in authors."
```

トリガは次のとおりです。

```
create trigger title_utrg
on titles
for update as
begin
     declare @num_updated int,
             @col1_var varchar(20),
             @col2_var varchar(20)
/* Determine how many rows were updated.*/
select @num_updated = @@rowcount
    if @num_updated = 0
    return
/* Ensure that title_id in titles is not changed.*/
if update (title_id)
    begin
      rollback transaction
     select @col1_var = title_id from inserted
     raiserror 35004, @col1_var
     return
   end
 /* Make sure dependencies to the publishers table are accounted for.*/
 if update(pub_id)
```

```
    begin
      if (select count(*) from inserted, publishers
          where inserted.pub_id = publishers.pub_id
          and inserted.pub_id is not null) != @num_updated
      begin
          rollback transaction
          select @col1_var = pub_id from inserted
          raiserror 35005, @col1_var
          return
      end
    end
/* If the column is null, raise error 24004 and rollback the
** trigger.If the column is not null, update the roysched table
** restricting the update.*/
    if update(price)
      begin
          if exists (select count(*) from inserted
          where price = null)
      begin
          rollback trigger with
          raiserror 24004 "Update failed : Price cannot be null."
      end
      else
      begin
          update roysched
          set lorange = 0,
          hirange = price * 1000
          from inserted
          where roysched.title_id =  inserted.title_id
      end
     end
end
```

最初のエラー・メッセージ 35004 (プライマリ・キー更新の失敗) をテストするには、次のように入力します。

```
update titles
set title_id = "BU7777"
where title_id = "BU2075"
```

2 番目のエラー・メッセージ 35005 (更新の失敗、オブジェクトの未検出) をテストするには、次のように入力します。

```
update titles
set pub_id = "7777"
where pub_id = "0877"
```

エラー・メッセージ 24004 (更新失敗、オブジェクトが null 値) を生成する 3 番目のエラーをテストするには、次のように入力します。

```
update titles
set price = 10.00
where title_id = "PC8888"
```

このクエリは、titles の price カラムが null なので、失敗します。このカラムが null でなければ、タイトルの PC8888 の価格を更新し、roysched テーブルについて必要な再計算を実行します。エラー 24004 は sysusermessages にはありませんが、この場合は有効です。これは、“rollback trigger with raiserror” というコード部分を示します。

外部キーの更新

外部キーを単独で変更または更新すると、エラーになります。外部キーは、プライマリ・キーのコピーです。両者のキーの間に存在する依存関係を損なわないようにしてください。外部キーを更新できるようにするには、**マスタ**・テーブルに対する更新をチェックし、プライマリ・キーと一致しなければその更新をロールバックするトリガを作成して、プライマリ・キーとの整合性を保護してください。

次の例では、トリガは、失敗の原因と考えられる title_id が salesdetail テーブルにないか、または titles テーブルにないかのいずれかの可能性についてテストします。

この例ではネストされた if...else 文を使用します。update 文の where 句にある値が salesdetail テーブルにある値と一致しない場合、1 番目の if 文は true です。つまり、inserted テーブルにはローが含まれておらず、select は null 値を返します。このテストが成功すると、inserted テーブル内の新しいローが titles テーブル内の title_id とジョインするかどうかが 2 番目の if 文によって確認されます。どのローもジョインしない場合は、トランザクションがロールバックされ、エラー・メッセージが表示されます。ジョインが成功すると、別のメッセージが表示されます。

```
create trigger forupdatetrig
on salesdetail
for update
as
declare @row int
/* Save value of rowcount.*/
select @row = @@rowcount
if update (title_id)
begin
    if (select distinct inserted.title_id
        from inserted) is null
      begin
      rollback transaction
        print "No, the old title_id must be in"
        print "salesdetail."
      end
    else
      if (select count(*)
          from titles, inserted
          where titles.title_id =
          inserted.title_id) != @row
      begin
```

```
            rollback transaction
            print "No, the new title_id is not in"
            print "titles."
        end
        else
          print "salesdetail table updated"
end
```

複数ローについての考慮事項

トリガの機能が再計算や進行中の計算値の出力である場合は、複数ローについての考慮事項が特に重要です。

計算値を管理するトリガには、group by 句、つまり暗黙のグループ化を実行するサブクエリが含まれています。これによって、複数のローが挿入、更新、削除されるときに、計算値が作成されます。group by 句ではオーバヘッドが余分にかかるため、以降の各例は、*@@rowcount* = 1 であるかどうか、つまり、トリガ・テーブルのローのうち 1 行だけに作用したかどうかをテストするように記述されています。*@@rowcount* = 1 の場合、group by 句がなくてもこのトリガは起動されます。

複数ローを使用した挿入トリガの例

次の挿入トリガによって、salesdetail ローが新しく追加されるごとに、titles テーブルの total_sales カラムが更新されます。salesdetail テーブルにローを追加して売り上げを記録すると、このトリガが起動されます。このトリガによって titles テーブルの total_sales カラムが更新され、total_sales カラムの値は、以前の値と salesdetail.qty を足した値に等しくなります。これによって、salesdetail.qty への挿入が合計に反映されます。

```
create trigger intrig
on salesdetail
for insert as
    /* check value of @@rowcount */
if @@rowcount = 1
    update titles
      set total_sales = total_sales + qty
      from inserted
      where titles.title_id = inserted.title_id
else
    /* when @@rowcount is greater than 1,
       use a group by clause */
    update titles
      set total_sales =
        total_sales + (select sum(qty)
      from inserted
```

```
      group by inserted.title_id
      having titles.title_id = inserted.title_id)
```

複数ローを使用した削除トリガの例

次に、1 つまたは複数の salesdetail ローが削除されるたびに titles テーブルの total_sales カラムを更新する、削除トリガの例を示します。

```
create trigger deltrig
on salesdetail
for delete
as
    /* check value of @@rowcount */
if @@rowcount = 1
    update titles
      set total_sales = total_sales - qty
      from deleted
      where titles.title_id = deleted.title_id
else
    /* when rowcount is greater than 1,
       use a group by clause */
    update titles
      set total_sales =
        total_sales - (select sum(qty)
      from deleted
      group by deleted.title_id
      having titles.title_id = deleted.title_id)
```

salesdetail テーブルからローが削除されると、このトリガが起動されます。このトリガによって titles テーブルの total_sales カラムが更新され、total_sales カラムの値は、salesdetail.qty から引かれた値を、前の値から引いた値と等しくなります。

複数ローを使用した更新トリガの例

次の更新トリガは、titles テーブルの total_sales カラムを、salesdetail の qty フィールドが更新されるたびに更新します (更新とは、削除後の挿入を指します)。このトリガは、inserted トリガ・テスト・テーブルと deleted トリガ・テスト・テーブルの両方を参照します。

```
create trigger updtrig
on salesdetail
for update
as
if update (qty)
begin
    /* check value of @@rowcount */
    if @@rowcount = 1
      update titles
```

```
            set total_sales = total_sales +
              inserted.qty - deleted.qty
            from inserted, deleted
            where titles.title_id = inserted.title_id
            and inserted.title_id = deleted.title_id
        else
        /* when rowcount is greater than 1,
           use a group by clause */
        begin
          update titles
            set total_sales = total_sales +
              (select sum(qty)
                from inserted
                group by inserted.title_id
                having titles.title_id =
                inserted.title_id)
          update titles
            set total_sales = total_sales -
              (select sum(qty)
                from deleted
                group by deleted.title_id
                having titles.title_id =
                deleted.title_id)
        end
    end
```

複数ローを使用した条件付き挿入トリガの例

受け入れられないデータ修正があるからといって、すべてのデータ修正をロールバックする必要はありません。トリガに相関サブクエリを指定すれば、修正されたローを個別に検査できます。「相関サブクエリの使用」(176 ページ) を参照してください。これによって、トリガはローごとに異なる動作を実行できます。

次のトリガの例では、junesales と呼ばれるテーブルを例にとります。次の文は、junesales テーブルの create 文です。

```
create table junesales
(stor_id    char(4)     not null,
ord_num     varchar(20) not null,
title_id    tid         not null,
qty         smallint    not null,
discount    float       not null)
```

条件付きのトリガをテストするには、junesales テーブルにローを 4 つ挿入してください。junesales ローのうちの 2 つには、titles テーブルにある title_id のどれにも一致しない ID があります。

```
insert junesales values ("7066", "BA27619", "PS1372", 75, 40)
insert junesales values ("7066", "BA27619", "BU7832", 100, 40)
insert junesales values ("7067", "NB-1.242", "PSxxxx", 50, 40)
```

```
insert junesales values ("7131", "PSyyyy", "PSyyyy", 50, 40)
```

junesales から salesdetail にデータを挿入するとき、文は次のようになります。

```
insert salesdetail
select * from junesales
```

conditionalinsert トリガは挿入をローごとに分析し、titles テーブル内に title_id がないローを削除します。

```
create trigger conditionalinsert
on salesdetail
for insert as
if
(select count(*) from titles, inserted
where titles.title_id = inserted.title_id)
    != @@rowcount
begin
  delete salesdetail from salesdetail, inserted
    where salesdetail.title_id = inserted.title_id
    and inserted.title_id not in
    (select title_id from titles)
  print "Only records with matching title_ids
    added."
end
```

トリガ・テストは、不必要なローを削除します。挿入されたばかりのローを削除する機能は、トリガが起動されるときに処理が起こる順序と同じです。つまり、当該テーブルと inserted テーブルにローが挿入されてから、トリガが起動されます。

トリガのロールバック

トリガをロールバックするには、rollback trigger 文または rollback transaction 文 (トリガがトランザクションの一部として起動される場合) のどちらかを使用できます。ただし、rollback trigger によってロールバックされるのは、トリガの影響とトリガを起動させた文だけです。rollback transaction は、トランザクション全体をロールバックします。次に例を示します。

```
begin tran
insert into publishers (pub_id) values ("9999")
insert into publishers (pub_id) values ("9998")
commit tran
```

2 番目の insert 文が publishers のトリガに rollback trigger を実行させると、2 番目の insert 文のみがロールバックされ、1 番目の insert 文はロールバックされません。代わりにそのトリガに rollback transaction 文を実行させると、insert 文は 2 つともトランザクションの一部としてロールバックされます。

rollback trigger の構文は次のとおりです。

```
rollback trigger
    [with raiserror_statement]
```

rollback transaction の構文については、「第 23 章 トランザクション：データの一貫性およびリカバリ」を参照してください。

raiserror_statement は、ユーザ定義のエラー・メッセージを出力したり、エラー状態が発生したことを記録するシステム・フラグを設定したりします。この文は、エラー状態のトランザクションのステータスがロールバックを反映するように、rollback trigger 文の実行時にクライアントにエラーを表示します。次に例を示します。

```
rollback trigger with raiserror 25002
    "title_id does not exist in titles table."
```

raiserror の詳細については、「第 15 章 バッチおよびフロー制御言語の使用」を参照してください。

挿入トリガの次の例は、「挿入トリガの例」(613 ページ) で説明した forinsertrig1 トリガと同様のタスクを実行します。ただし、このトリガは、rollback transaction の代わりに rollback trigger を使用して、トランザクションではなく、挿入がロールバックされるときにエラーが発生します。

```
create trigger forinsertrig2
on salesdetail
for insert
as
if (select count(*) from titles, inserted
    where titles.title_id = inserted.title_id) !=
    @@rowcount
   rollback trigger with raiserror 25003
     "Trigger rollback: salesdetail row not added
      because a title_id does not exist in titles."
```

rollback trigger 文が実行されると、Adaptive Server は現在実行中のコマンドをアボートし、トリガ内の残りのコマンドの実行も停止します。rollback trigger 文を実行するトリガがほかのトリガ内にネストされている場合は、Adaptive Server は、最初のトリガを起動させた更新を含め、それまでにこれらのトリガで実行された作業をすべてロールバックします。

rollback transaction 文を含むトリガがバッチから実行されるときは、そのトリガによってバッチ全体がアボートされます。次の例で、insert 文によって (forinsertrig1 などの) rollback transaction 文を含むトリガが起動される場合は、バッチ全体がアボートされるため delete 文は実行されません。

```
insert salesdetail values ("7777", "JR123",
    "PS9999", 75, 40)
delete salesdetail where stor_id = "7067"
```

rollback transaction 文を含むトリガが「ユーザ定義トランザクション」から起動される場合は、rollback transaction 文によってバッチ全体がロールバックされます。次の例では、insert 文によって rollback transaction 文を含むトリガが起動されると、update 文もロールバックされます。

```
begin tran
update stores set payterms = "Net 30"
    where stor_id = "8042"
insert salesdetail values ("7777", "JR123",
    "PS9999", 75, 40)
commit tran
```

ユーザ定義トランザクションの詳細については、「第 23 章 トランザクション：データの一貫性およびリカバリ」を参照してください。

Adaptive Server は、トリガの外で実行される rollback trigger 文を無視し、その文に関連する raiserror も発行しません。ただし rollback trigger 文は、トリガの外であってもトランザクション内で実行されれば、エラーを生成します。このエラーによって Adaptive Server はトランザクションをロールバックし、現在の文のバッチをアボートします。

グローバル・ログイン・トリガ

グローバル・ログイン・トリガを設定するには、sp_logintrigger を使用します。これは、ユーザのログインごとに実行されます。ユーザ固有のアクションを取得するには、sp_modifylogin または sp_addlogin を使用してユーザ固有のログイン・トリガを設定します。

注意 トレース・フラグ -T4073 を設定して、sp_logintrigger をアクティブ化できます。

『リファレンス・マニュアル：プロシージャ』を参照してください。

トリガのネスト

トリガは 16 レベルの深さまでネストできます。現在のネスト・レベルは、*@@nestlevel* グローバル変数に格納されます。ネストは、インストール時には使用可能になっています。システム管理者は、設定パラメータ allow nested triggers を使用すると、トリガのネストのオンとオフを切り替えられます。

ネスト・トリガが使用可能な場合は、2 番目のトリガを含むテーブルを変更するトリガによってその 2 番目のトリガが起動され、この 2 番目のトリガによって 3 番目のトリガが起動され、以下同様に続きます。このチェーンになっているトリガの 1 つで無限ループが発生した場合は、ネスト・レベルが超過し、トリガはアボートされます。ネスト・トリガは、直前のトリガが作用したローのバックアップ・コピーを保管するなどのハウスキーピング機能を実行する場合に便利です。

たとえば、delcascadetrig トリガによって削除された titleauthor ローのバックアップ・コピーを保存するトリガを titleauthor に作成できます。delcascadetrig トリガが動作している状態で、titles テーブルから title_id “PS2091” を削除すると titleauthor からも対応するすべてのローが削除されます。このデータを保存するには、削除されたデータを別の del_save テーブルに保管する delete トリガを titleauthor に作成します。

```
create trigger savedel
on titleauthor
for delete
as
insert del_save
select * from deleted
```

順序が重要なシーケンスで、ネストされたトリガを使用することをおすすめします。データ修正をカスケードするには、この章で前述した delcascadetrig のように別々のトリガを使用してください。この説明は、「カスケード型削除の例」(614 ページ) にあります。

注意　トリガをトランザクション内に設定した場合、ネスト・トリガのどのレベルで障害が発生しても、トランザクションは取り消され、すべてのデータ修正はロールバックされます。どこで障害が発生したかを判別するには、トリガ内で print または raiserror を使用します。

どのネスト・レベルでも、トリガの rollback transaction 文は各トリガの結果をロールバックし、トランザクション全体を取り消します。rollback trigger 文は、ネスト・トリガと最初のトリガを起動させるデータ修正文にだけ作用します。

トリガの自己再帰

デフォルトでは、トリガは自分自身を再帰的に呼び出すことはできません。たとえば、更新トリガは、そのトリガ内で同じテーブルに対する 2 回目の更新をするために、自分自身を呼び出すことはできません。あるテーブルの 1 つのカラムの更新トリガによって別のカラムが更新される場合、更新トリガは 1 回だけ起動されます。ただし、set コマンドの self_recursion オプションを使用すると、トリガは自分自身を再帰的に呼び出すことができます。allow nested triggers 設定変数も有効にして、自己再帰が使用できるようにする必要があります。

self_recursion の設定が有効なのは、現在のクライアント・セッションが実行されている間だけです。このオプションがトリガの一部として設定されている場合、その効果は、オプションを設定するトリガの範囲に限定されます。self_recursion オプションを on に設定するトリガが戻されるか、またはこのトリガが別のトリガを起動させると、このオプションは off になります。トリガが self_recursion オプションをオンに設定すると、トリガ内に自分自身を再び起動させる動作が指定された場合に自分自身を繰り返しループできますが、ネスト・レベルは 16 を超えることはできません。

たとえば、pubs2 に次のような new_budget テーブルがあるとします。

```
select * from new_budget

unit            parent_unit     budget
--------------- --------------- -------
one_department  one_division    10
one_division    company_wide    100
company_wide    NULL            1000

(3 rows affected)
```

次のトリガを作成して、budget カラムが変更されると new_budget を再帰的に更新するようにできます。

```
create trigger budget_change
on new_budget
for update as
if exists (select * from inserted
            where parent_unit is not null)
begin
    set self_recursion on
    update new_budget
    set new_budget.budget = new_budget.budget +
        inserted.budget - deleted.budget
    from inserted, deleted, new_budget
    where new_budget.unit = inserted.parent_unit
        and new_budget.unit = deleted.parent_unit
end
```

one_department の予算を 3 だけ増加して new_budget.budget を更新すると、Adaptive Server は次のように動作します (ネスト・トリガが使用可能になっていることを前提とします)。

1 one_department を 10 から 13 に増やすと、budget_change トリガが起動されます。

2 起動されたトリガによって、one_department の親 (この場合 one_division) が 100 から 103 に更新され、これによって再びトリガが起動されます。

3 起動されたトリガによって、one_division の親 (この場合 company_wide) が 1000 から 1003 に更新され、これによってトリガの 3 回目の起動が行われます。

4　起動されたトリガは、company_wide の親を更新しようとしますが、そこには何も存在しない (値は "NULL" です) ために最後の update は発行されず、トリガは起動されずに自己再帰を終了します。最終結果を参照するために、new_budget に対して次のように問い合わせます。

```
select * from new_budget

unit            parent_unit     budget
--------------- --------------- -------
one_department  one_division    13
one_division    company_wide    103
company_wide    NULL            1003

(3 rows affected)
```

他の方法を使用しても、再帰的にトリガを実行できます。トリガはストアド・プロシージャを呼び出し、それがトリガを再び起動させる動作を起こします (トリガは、ネスト・トリガが使用可能な場合にだけ再起動されます)。再帰の回数を制限する条件がトリガ内にない場合は、ネスト・レベルをオーバフローできます。

たとえば、更新を実行するストアド・プロシージャが更新トリガによって呼び出されるとき、allow nested triggers が off に設定されている場合、トリガおよびストアド・プロシージャは一度だけ実行されます。nested triggers が on に設定されていて、更新の回数がトリガまたはストアド・プロシージャ内のある条件によって 16 を超えた場合、このループはトリガまたはプロシージャが最大ネスト値の 16 レベルを超えるまで続きます。

トリガに関する規則

複数ロー・データの修正、トリガのロールバック、トリガのネストの結果に加え、トリガを記述するときに考慮するべき注意事項があります。

トリガとパーミッション

トリガは、テーブル上に定義されます。テーブルで create trigger や drop trigger を実行するパーミッションが与えられているのは、そのテーブルの所有者だけであり、他のユーザに譲渡することはできません。

Adaptive Server は、パーミッションを持っていない動作を実行するトリガの定義を受け入れます。このようなトリガが存在すると、トリガ・テーブルの変更はアボートします。これは、パーミッションが不正なため、トリガが起動しても失敗するからです。このトランザクションは取り消されます。パーミッションを修正するか、トリガを削除してください。

たとえば、Jose が、salesdetail テーブルを所有しており、そこにトリガを作成します。salesdetail.qty が更新されると titles.total_sales が更新されるようにトリガが設計されているとします。しかし、titles テーブルの所有者である Mary は、titles テーブルのパーミッションを Jose に付与していません。Jose は salesdetail テーブルを更新しようとしますが、Adaptive Server は、トリガだけでなく、titles テーブルに Jose のパーミッションがないことも検出すると、更新トランザクションをロールバックします。Jose は、titles.total_sales 上での更新パーミッションを Mary に付与してもらうか、salesdetail テーブルのトリガを削除する必要があります。

トリガの制約

Adaptive Server では、トリガについて次のような制限があります。

- 1 つのテーブルには、insert、update、および delete のそれぞれを 1 つずつ、最大 3 つのトリガを定義できます。
- 各トリガを適用できるテーブルは 1 つだけである。ただし、1 つのトリガに、update、insert、delete の 3 つすべてのユーザ操作を組み合わせられる。
- ビューまたはセッション固有のテンポラリ・テーブル上にトリガを作成できないが、トリガがビューまたはテンポラリ・テーブルを参照することはできる。
- writetext 文は、挿入トリガまたは更新トリガをアクティブにしない。
- truncate table 文はローをすべて削除するため、where 句のない delete 文に似ているが、個々のローの削除を記録していないため、トリガを起動できない。
- テンポラリ・オブジェクト (@object) 上に、トリガ、インデックス、またはビューを作成できない。
- システム・テーブルにはトリガを作成できない。システム・テーブルにトリガを作成しようとすると、Adaptive Server はエラー・メッセージを返し、トリガの作成を取り消す。
- 挿入または削除を行う同じトリガのあるテーブルの text カラムまたは image カラムから選択するトリガは使用できない。
- コンポーネント統合サービスが有効の場合、inserted や deleted テーブルを介して挿入、更新、または削除中のローを検査できないので、プロキシ・テーブルでのトリガの使用は限られる。プロキシ・テーブルにトリガを作成して呼び出すことはできる。ただし、insert はリモート・サーバに渡されるため、削除または挿入されたデータは、プロキシ・テーブルのトランザクション・ログに記録されない。したがって、inserted および deleted テーブルがトランザクション・ログへの実際のビューであり、それらのテーブルにプロキシ・テーブルのデータは含まれない。

暗黙的および明示的な null 値

カラムが select リストまたは values 句の中で値を割り当てられるときは、常に insert 文に対する if update(*column_name*) 句は、文が true になります。「明示的な null」またはデフォルトによって、値がカラムに割り当てられ、トリガが起動されます。「暗黙的な null」を使用しても、カラムに値は割り当てられず、トリガも起動されません。

たとえば、このテーブルを作成するとします。

```
create table junk
(a int null,
 b int not null)
```

このトリガを記述します。

```
create trigger junktrig
on junk
for insert
as
if update(a) and update(b)
        print "FIRING"

    /*"if update" is true for both columns.
      The trigger is activated.*/
insert junk (a, b) values (1, 2)

    /*"if update" is true for both columns.
      The trigger is activated.*/
insert junk values (1, 2)

    /*Explicit NULL:
      "if update" is true for both columns.
      The trigger is activated.*/
insert junk values (NULL, 2)

    /* If default exists on column a,
      "if update" is true for either column.
      The trigger is activated.*/
insert junk (b) values (2)

    /* If no default exists on column a,
      "if update" is not true for column a.
      The trigger is not activated.*/
insert junk (b) values (2)
```

この句を使用するだけで、上記とまったく同じ結果になります。

```
if update(a)
```

暗黙の null の挿入を許可しないトリガを作成するには、次の文を使用します。

```
if update(a) or update(b)
```

これで、トリガ内の SQL 文は、*a* または *b* が null であるかどうかをテストできます。

トリガとパフォーマンス

パフォーマンスについては、トリガのオーバヘッドは通常、非常に低いものです。トリガの実行にかかる時間のほとんどは、メモリかデータベース・デバイスのどちらかにある他のテーブルを参照するのに費やされます。

deleted と inserted トリガ・テスト・テーブルは、常にアクティブなメモリに存在します。トリガに参照される他のテーブルの位置によって、オペレーションにかかる時間の合計が決まります。

トリガの *set* コマンド

トリガ内で set コマンドを使用できます。呼び出した set オプションは、トリガの実行中は有効です。実行後は以前の設定に戻ります。

名前変更とトリガ

トリガに参照されるオブジェクト名を変更する場合は、トリガを削除してから再作成し、参照するオブジェクトの新しい名前を、このトリガのテキストに反映させてください。トリガが参照するオブジェクトのレポートを取得するには、sp_depends を使用します。最も確実は方法は、トリガに参照されるテーブルやビューの名前を変更しないことです。

トリガに関するヒント

トリガを作成するときは、これらの点を考慮してください。

- ベース・テーブルを更新するストアド・プロシージャを呼び出すための insert または update トリガを設定するとします。allow nested triggers 設定パラメータが true に設定されていると、トリガは無限ループに入ります。insert または update トリガを実行する場合は、その前に sp_configure “nested triggers” を false に設定してください。
- drop table を実行すると、そのテーブルに従属するすべてのトリガも削除されます。このようなトリガを残す場合は、テーブルを削除する前に、sp_rename を使用してトリガ名を変更します。
- トリガで指定されているテーブルおよびビューについてのレポートを参照するには、sp_depends を使用します。
- トリガの名前を変更するには、sp_rename を使用します。

- トリガは、クエリ 1 つにつき一度だけ起動されます。クエリがループ内で繰り返されると、そのたびにトリガも起動します。

トリガの無効化

insert、update、delete コマンドは、通常は検出したトリガを起動するため、操作の実行に必要な時間が増大します。バルク insert、update、または delete オペレーション中にトリガを無効にするには、alter table コマンドの disable trigger オプションを使用できます。このオプションでは、テーブルに関連するすべてのトリガを無効にすることも、特定のトリガを無効にすることもできます。ただし、コピーが完了すると、無効にしたどのトリガも起動されます。

bcp は、データのロードを最高速度で行えるよう、ルールとトリガを起動しません。

ルールおよびトリガに違反するローを見つけるには、データをテーブルにコピーし、ルールまたはトリガの条件をテストするクエリやストアド・プロシージャを実行してください。

alter table... disable trigger は、次の構文を使用します。

```
alter table [database_name.[owner_name].]table_name
    {enable | disable } trigger [trigger_name]
```

table_name はトリガを無効にするテーブルの名前で、*trigger_name* は無効にするトリガの名前です。たとえば、pubs2 データベースにある del_pub トリガを無効にするには、次のようにします。

```
alter table pubs2
disable del_pubs
```

トリガ名を指定しないと、alter table はテーブルに定義されたすべてのトリガを無効にします。

データベースのロードが完了した後トリガを再度有効にするには、alter table... enable trigger を使用します。del_pub トリガを再度有効にするには、次の文を発行します。

```
alter table pubs2
enable del_pubs
```

注意　トリガを無効にする機能を使用できるのは、テーブルの所有者またはデータベース管理者だけです。

トリガに insert 文、update 文、または delete 文が含まれている場合、それらの文はトリガが無効になっている間は実行されないため、テーブルの参照整合性に影響を与えます。

トリガの削除

トリガを除去するには、トリガを削除するか、またはトリガが関連付けられているトリガ・テーブルを削除します。

drop trigger 文の構文は次のとおりです。

```
drop trigger [owner.]trigger_name
    [, [owner.]trigger_name]...
```

テーブルを削除すると、Adaptive Server は、そのテーブルに関連付けられたトリガも削除します。drop trigger パーミッションは、デフォルトではトリガ・テーブルの所有者に付与されており、他のユーザには譲渡できません。

トリガに関する情報の取得

トリガは、データベース・オブジェクトとして sysobjects テーブルに名前がリストされています。sysobjects テーブルの type カラムは “TR” という省略記号によってトリガを識別します。次のクエリによって、データベースに存在するトリガが検出されます。

```
select *
from sysobjects
where type = "TR"
```

各トリガのソース・テキストは、syscomments に保管されています。トリガの実行プランは、sysprocedures に保管されます。次の項で説明するシステム・プロシージャは、システム・テーブルからトリガについての情報を提供します。

sp_help

sp_help を使用すると、トリガのレポートを得ることができます。たとえば、次のようにして、deltitle についての情報を取得できます。

```
sp_help deltitle

Name        Owner   Type
----------- ------- -----------
deltitle    dbo     trigger
Data_located_on_segment  When_created
-----------------------  ------------------
not applicable           Jul 10 1997 3:56PM

(return status = 0)
```

また、sp_help を使用して、トリガの有効または無効に関するステータスを確認できます。

```
1> sp_help trig_insert
2> go
Name Owner
Type
-----------------------------------
trig_insert dbo
trigger
(1 row affected)
data_located_on_segment When_created
-----------------------------------
not applicable Aug 30 1998 11:40PM
Trigger enabled
(return status = 0)
```

sp_helptext

トリガのソース・テキストを表示するには、次のようにシステム・プロシージャ sp_helptext を実行します。

```
sp_helptext deltitle

# Lines of Text
---------------
              1
text
--------------------------------------------
create trigger deltitle
on titles
for delete
as
if (select count(*) from deleted, salesdetail
where salesdetail.title_id = deleted.title_id) >0
begin
      rollback transaction
      print "You can't delete a title with sales."
end
```

トリガのソース・テキストが sp_hidetext を使用して暗号化されている場合、Adaptive Server はテキストが隠されていることを通知するメッセージを表示します。『リファレンス・マニュアル：プロシージャ』を参照してください。

システム・セキュリティ担当者が、Adaptive Server を評価済み設定で実行するために sp_configure の allow select on syscomments.text column パラメータをリセットした場合、トリガの作成者またはシステム管理者のみが、sp_configure を使用してトリガのソース・テキストを表示できます。『システム管理ガイド第 1 巻』の「第 12 章 セキュリティの概要」を参照してください。

sp_depends

sp_depends によって、オブジェクトを参照するトリガ、またはトリガが作用するすべてのテーブルまたはビューがリストされます。次の例では、トリガ deltitle によって参照されるオブジェクトをすべてリストする sp_depends の使用方法を示します。

```
sp_depends deltitle

Things the object references in the current database.
object              type       updated  selected
----------------    ----------  ------- --------
dbo.salesdetail     user table  no      no
dbo.titles          user table  no      no
(return status = 0)
```

次の文によって、salesdetail テーブルを参照するオブジェクトがすべてリストされます。

```
sp_depends salesdetail
Things inside the current database that reference the object.
object                         type
---------------------------  ----------------
dbo.deltitle                 trigger
dbo.history_proc             stored procedure
dbo.insert_salesdetail_proc  stored procedure
dbo.totalsales_trig          trigger
(return status = 0)
```

第 21 章

ロー内/ロー外の LOB

概要

Adaptive Server では、ページ内の空き領域に応じて、text、image、および unitext データ型の小さなロー内の LOB カラムの保存をサポートしています。LOB のサイズが拡大するか、空き容量が他のロー内のカラム (varchar や varbinary データ型に使用されるカラムなど) に使用されている場合、Adaptive Server ではロー内の LOB データがロー外の記憶領域に連続的にマイグレートされ、データがロー内テキスト・ポインタに自動的に置換されます。

次のことを実行することができます。

- create table による LOB カラムのロー内の記憶領域の指定
- alter table による LOB カラムの保存方法の変更
- create または alter database コマンドによるデータベース全体での LOB カラムのロー内の長さの管理

Component Integration Services (CIS) プロキシ・テーブルをロー内 LOB カラムを含むリモート Adaptive Server テーブルにマップする場合、このプロキシ・テーブル内の LOB カラムをロー外 LOB として定義する必要があります。どのデータ転送も、ロー外 LOB カラム・データとして発生します。

15.7 より前のバージョンの Adaptive Server では、ラージ・オブジェクト (LOB) カラム (text、image、unitext、XML など) のロー外をテキスト・ポインタ (txtptr) を使用して保存して、開始ページ ID 値を特定し、ポインタをデータ・ローに保存していました。これには、直列化 Java クラスが含まれます。このクラスがロー内に保管されるのは、これらのクラスが固定最大長より短い場合だけです。

ロー内 LOB カラムの圧縮

ロー・レベルまたはページ・レベルの圧縮をセットアップしても、ロー内 LOB カラム (またはそのメタデータ) は圧縮されません。ただし、LOB カラムを圧縮向けに定義した場合、LOB データがロー外になり、ログ・データ・サイズが論理ページ・サイズを超えると、LOB 圧縮がロー内 LOB のデータ部分に拡張されます。

ロー外 LOB カラムと同様に、さまざまなロー内 LOB カラムのデータは、LOB データがロー外になった場合、さまざまな LOB 圧縮レベルで圧縮することができます。しかし、Adaptive Server がロー外 LOB カラムのテキスト・ポインタ・フィールドを圧縮することはありません。

with lob_compression 句を使用して、テーブル内の LOB カラムすべての圧縮レベルを設定する場合、必ず “not compressed” カラム・レベル句を使用して、圧縮しない個別のロー内 LOB カラムまたはその他の LOB カラムを指定してください。

ロー内記憶領域を使用するためのロー外 LOB データのマイグレート

テーブル全体で機能するユーティリティ操作および一部のデータ定義言語 (data definition language：DDL) では、コピー先スキーマのデータ・ローと一致するようにデータ・ローを書き直すことにより、テーブル・データをすべてコピーします。ロー内記憶領域を使用するために LOB を変換する方法は、いくつかあります。

- update set *column* = *column*
- alter table スキーム変更 (add not null、modify データ型または null 入力可能性、あるいは drop column － ローのコンテンツとレイアウト) は、スキーム変更に対応するよう再整理される。ロー全体が再構築される。

  alter table ... partition by は、データ・ローをさまざまなパーティションに分散させることで、分割スキームを変更する。ローは、データ・コピーの一部として再フォーマットされる。ただし、この操作でテーブル・スキームが変更されることはない。
- reorg rebuild により、ローがデータ移動の一部として再構築される。
- bcp バルク・コピー・ユーティリティでは、ロー内 LOB カラムを使用するテーブルをサポートする。「ロー内カラムとバルク・コピー」を参照。

これらのうちいずれかの方法で既存データをマイグレートすることができます。その際、テキスト・ページの領域使用量を削減し、ロー内 LOB 記憶領域に移動することができます。

ロー内カラムとバルク・コピー

Adaptive Server の bcp ユーティリティでは、ロー内 LOB カラムを使用するテーブルをサポートしています。そして、ロー外データが定義済みロー内サイズに適合し、結果として得られるローがページ・サイズ制限を満たす限り、LOB カラムを保存します。

さらに、bcp in は、同ユーティリティが char および varchar のデータ型変換を処理するのと同じ方法でロー内 LOB として保存される場合、text データ型の文字セット変換を処理します。

つまり、サーバ側文字セット変換がアクティブの場合、変換後の必要領域が元の長さと異なると、Adaptive Server はロー内 LOB データを拒否します。これが発生すると、次が表示されます。

```
BCP insert operation is disabled when data size is
changing between client and server character sets.
Please use BCP's -Y option to invoke client-side
conversion.
```

そのため、bcp -Y オプションを使用して Adaptive Server に強制的にサーバ上ではなくクライアント上で文字セット変換を実行させることをお勧めします。これにより、クライアントとサーバの両方の文字データの長さを確実に同じにします。

既存データをマイグレートする各種メソッドの例

以降の例では、ロー外 LOB データをロー内記憶領域にマイグレートするために使用するメソッドごとに異なる構文を示し、結果を比較します。それらの結果は非常によく似ているため、どのメソッドを選択すべきかは、状況と好みで決まります。

どの例でも、select into を使用して、元のテーブル mymsgs のコピーを作成します。すると、コピーされたテーブルの description カラムのロー内長さは変更されます。説明中のメソッドを使用して、ロー外 LOB をロー内記憶領域にマイグレートします。元のテーブルと変更後のコピーとの間で記憶域使用量を比較し、LOB 記憶領域が大幅に減少したことを示します。

mymsgs 例テーブルの設定

この例では、pubs2 データベースから mymsgs テーブルを作成します。カラム・コンテンツをロー外からロー内の記憶領域にマイグレートするための準備として、description カラムに varchar ではなく text を指定します。

```
1> use pubs2
2> go
1> exec sp_drop_object mymsgs, 'table'
2> go
1> create table mymsgs (

          error                int              not null
        , severity             smallint         not null
        , dlevel               smallint         not null
```

```
        , description          text
        , langid               smallint          null
        , sqlstate             varchar (5)       null

) lock datarows
2> go
1> insert mymsgs select * from master..sysmessages
2> go
(9564 rows affected)

1> exec sp_spaceusage display, 'table', mymsgs
2> go
All the page counts in the result set are in the unit 'KB'.
 OwnerName TableName IndId  NumRows UsedPages RsvdPages ExtentUtil ExpRsvdPages
PctBloatUsedPages  PctBloatRsvdPages
 --------- --------- ----- ------- --------- --------- ---------- ------------ -----
----------------- ------------------
 dbo       mymsgs       0    9564.0    372.0     384.0      96.87        320.0
           16.25             20.00
 dbo       mymsgs     255    NULL    19132.0   19140.0      99.95      19136.0
           00                 0.02


1> use pubs2
1>
2> dump tran pubs2 with no_log
1>
2> /* 各実行の前に、spaceusage stats テーブルを削除します */
3> exec sp_drop_object spaceusage_object, 'table'
Dropping table spaceusage_object
(return status = 0)

1>
2> exec sp_spaceusage archive, 'table', mymsgs
Data was successfully archived into table 'pubs2.dbo.spaceusage_object'.
(return status = 0)
```

update 文を使用したマイグレート

この例では、update set *column* = *column* を使用して、ロー外 LOB をロー内記憶領域にマイグレートします。select into コマンドは、最初に mymsgs のコピーをそのロー外データも含めて mymsgs_test_upd に作成し、その過程で、ロー外データをロー内に移動します。すると、update コマンドを移動して、ロー外 LOB をロー内記憶領域に再配置することができるようになります。

```
1> exec sp_drop_object mymsgs_test_upd, 'table'
Dropping table mymsgs_test_upd
(return status = 0)
1>
2> select * into mymsgs_test_upd from mymsgs
(9564 rows affected)
1>
2> exec sp_spaceusage display, 'table', mymsgs_test_upd
All the page counts in the result set are in the unit 'KB'.
```

```
 OwnerName TableName      IndId NumRows UsedPages RsvdPages ExtentUtil ExpRsvdPages
    PctBloatUsedPages PctBloatRsvdPages
 --------- -------------- ----- ------- --------- --------- ---------- ------------
    ----------------- -----------------
dbo       mymsgs_test_upd    0  9564.0     318.0     320.0     99.37         272.0
                22.31             17.65
dbo       mymsgs_test_upd  255    NULL   19132.0   19136.0     99.97       19136.0
                 0.00              0.00

(1 row affected)
(return status = 0)
```

mymsgs_test_upd の領域使用量は、mymsgs テーブルの場合とほぼ同じです。ロー外 LOB では、約 19KB の記憶領域を消費します。

```
)
1> alter table mymsgs_test_upd modify description in row (300)
1> sp_spaceusage
2> go
2> update mymsgs_test_upd set description = description
(9564 rows affected)
1>
2> exec sp_spaceusage display, 'table', mymsgs_test_upd
All the page counts in the result set are in the unit 'KB'.

 OwnerName TableName      IndId NumRows UsedPages RsvdPages ExtentUtil ExpRsvdPages
    PctBloatUsedPages PctBloatRsvdPages
 --------- -------------- ----- ------- --------- --------- ---------- ------------
    ----------------- -----------------
dbo       mymsgs_test_upd    0  9564.0    1246.0    1258.0      99.04         272.0
               379.23            362.50
dbo       mymsgs_test_upd  255    NULL       6.0      32.0     18.75          16.0
                 0.00            100.00

(1 row affected)
(return status = 0)
1>
2> exec sp_spaceusage archive, 'table', mymsgs_test_upd
Data was successfully archived into table 'pubs2.dbo.spaceusage_object'.
(return status = 0)
```

データ・レイヤ indid=0 の RsvdPages のサイズは、変更されています。以前には 320KB でしたが、今では 1258KB です。その一方で、LOB カラム indid=255 の予約ページのサイズは、19,136KB から約 32KB に減少し、ロー外記憶領域がロー内に変化したことが示されます。

注意　テーブルが非常に大きい (ロー数が 100 万を超えるなど) 場合、update 文の実行には長時間かかることがあります。where 句を使用して一度に少数のローを選択する場合は、必ずキー・インデックスを使用してテーブル内のローすべてを特定して、変換時にどのローも見逃さないようにします。

---

reorg rebuild の使用

この例では、reorg rebuild を使用してデータ移動の一部としてローを再構築し、ロー内 LOB を保存できるように mymsgs_test_reorg を再構築します。

```
1> exec sp_drop_object mymsgs_test_reorg, 'table'
Dropping table mymsgs_test_reorg
(return status = 0)
1>
2> select * into mymsgs_test_reorg from mymsgs
(9564 rows affected)

1> alter table mymsgs_test_reorg modify description in row (300)
1>
2> REORG REBUILD mymsgs_test_reorg
Beginning REORG REBUILD of table 'mymsgs_test_reorg'.
(9564 rows affected)
REORG REBUILD of table 'mymsgs_test_reorg' completed.
1>
2> exec sp_spaceusage display, 'table', mymsgs_test_reorg
All the page counts in the result set are in the unit 'KB'.
 OwnerName TableName      IndId NumRows UsedPages RsvdPages ExtentUtil ExpRsvdPages
    PctBloatUsedPages PctBloatRsvdPages
 --------- -------------- ----- ------- --------- --------- ---------- ------------
    ----------------- -----------------
dbo       mymsgs_test_reorg   0  9564.0    1230.0    1242.0     99.03         272.0
              373.08            356.62
dbo       mymsgs_test_reorg 255    NULL       6.0      32.0     18.75          16.0
                0.00              0.00

(1 row affected)
(return status = 0)
1>
2> exec sp_spaceusage archive, 'table', mymsgs_test_reorg
Data was successfully archived into table 'pubs2.dbo.spaceusage_object'.
(return status = 0)
```

データ・コピーでの alter table を使用したマイグレート

この例では、alter table を使用してあるカラムを削除し、新規カラムとして追加し直します。これは、ロー・コンテンツにより本質的には未変化のまま、description カラムが変更されるようにするためです。この例では、データ・コピー (drop カラムや add not null カラムなど) が必要とされることがある alter table スキーム変更操作の副作用として、LOB カラムをロー内記憶領域に移動する方法を示します。

```
1> exec sp_drop_object mymsgs_test_alttab, 'table'
Dropping table mymsgs_test_alttab
(return status = 0)
1>
2> select * into mymsgs_test_alttab from mymsgs
(9564 rows affected)

1> alter table mymsgs_test_alttab modify description in row (300)
```

```
1>
2> alter table mymsgs_test_alttab
3> DROP dlevel
4> ADD newdlevel int default 0 not null
(9564 rows affected)
1>
2> exec sp_spaceusage display, 'table', mymsgs_test_alttab
Warning: Some output column values in the result set may be incorrect.Running 'update
statistics' may help correct them.
All the page counts in the result set are in the unit 'KB'.
 OwnerName TableName      IndId NumRows UsedPages RsvdPages ExtentUtil ExpRsvdPages
    PctBloatUsedPages PctBloatRsvdPages
 --------- -------------- ----- ------- --------- --------- ---------- ------------
    ----------------- -----------------
dbo       mymsgs_test_alttab   0 9564.0    1252.0    1258.0      99.52        1728.0
              -27.46            -27.20
dbo       mymsgs_test_alttab 255   NULL       6.0      32.0      18.75          16.0
                0.00            100.00


(1 row affected)
(return status = 0)
1>
2> exec sp_spaceusage archive, 'table', mymsgs_test_alttab
Warning: Some output column values in the result set may be incorrect.Running 'update
statistics' may help correct them.
Data was successfully archived into table 'pubs2.dbo.spaceusage_object'.
(return status = 0)
```

これは概要レポートであり、これらの例で使用されているテーブルの領域使用量情報を示しており、使用領域が大幅に減少していることが分かります。LOB カラム (indid=255) の元の mymsgs テーブルでは、RsvdPages のサイズは 19,140 KB でしたが、この領域は、3 つすべてのサンプル・テーブルで 95% 以上減少しています。

```
1> exec sp_spaceusage report, 'table', 'mymsgs%', 'OwnerName, TableName, IndId,
NumRows, RsvdPages, UsedPages, ExtentUtil'
All the page counts in the result set are in the unit 'KB'.
 OwnerName TableName          IndId NumRows RsvdPages UsedPages ExtentUtil
 --------- ------------------ ----- ------- --------- --------- ----------
 dbo       mymsgs                 0  9564.0     318.0     318.0     100.00
 dbo       mymsgs               255    NULL   19140.0   19132.0      99.95
 dbo       mymsgs_test_alttab     0  9564.0    1258.0    1252.0      99.52
 dbo       mymsgs_test_alttab   255    NULL      32.0       6.0      18.75
 dbo       mymsgs_test_reorg      0  9564.0    1242.0    1230.0      99.03
 dbo       mymsgs_test_reorg    255    NULL      32.0       6.0      18.75
 dbo       mymsgs_test_upd        0  9564.0    1258.0    1246.0      99.04
 dbo       mymsgs_test_upd      255    NULL      32.0       6.0      18.75

(1 row affected)
(return status = 0)
```

ロー内 LOB 長を選択するためのガイドライン

ロー内 LOB 長の選択は、データ・ページに使用される記憶領域、LOB ページ、および 1 データ・ページにちょうど収まるローの数に影響を及ぼします。

- 論理ページ・サイズより大きいロー内 LOB 長を指定することはできません。ロー内の記憶領域では、ページ・サイズより小さい LOB 値だけが考慮されるからです。反対に、非常に小さいロー内 LOB 値を指定すると、ロー内に移動される LOB カラム数は非常に少なくなるので、LOB 記憶領域の節約にはつながらないことがあります。
- 一般的なロー内 LOB 長は、LOB カラムの最短データ長と論理ページ・サイズとの間となります。ロー内長値が大きい場合、データ・ページ全体がわずか 1 ローでいっぱいになってしまうことがあります。そのため、都合の良い値としては、ロー外 LOB カラムの平均データ長当たりが現実的です。これなら、長さはページ・サイズ未満です。
- ロー内 LOB 長の選択は、多数のローを返し、LOB カラムを参照しないクエリのスキャン・パフォーマンスにも悪影響を及ぼす可能性があります。ロー内 LOB 値が大きいため、データ・ページにちょうど収まるローの数が非常に少ない場合、スキャンされるデータ・ページの数が非常に多数になるため、クエリ応答が遅くなってしまう可能性があります。

  テーブルのデータ長を調べて、ロー内 LOB 長を推定し、ページにちょうど収まるローの数がわずか 1 つや 2 つにならないようにします。バランス・パフォーマンスは、減少した LOB 記憶領域に影響を及ぼします。

❖ ロー内 LOB 長選択の特定

1 LOB がロー外に保存されている状態で、テーブルのデータ・ロー・サイズの最小値、最大値、および平均値を特定します。

```
1> select i.minlen, t.datarowsize, i.maxlen
2> from sysindexes i, systabstats t
3> where i.id = object_id('DYNPSOURCE')
4>   and i.indid in (0, 1)
5>   and i.id = t.id
6> go

 minlen datarowsize                maxlen
------  --------------------------- ------
 9                       105.000000    201
```

2 目的のロー外カラムの最小、平均、および最大データ長を計算します。

```
1> select datalength(FIELDINFO) as fieldinfo_len into
       #dynpsource_FIELDINFO
2> from DYNPSOURCE
3> where datalength(FIELDINFO) < @@maxpagesize
4> go
```

```
(65771 rows affected)
1> select minlen = min(fieldinfo_len), avglen = avg(fieldinfo_len),
       maxlen = max(fieldinfo_len)
2> from #dynpsource_FIELDINFO
3> go

 minlen     avglen      maxlen
----------- ----------- -----------
        536        7608       16080
```

DYPNSOURCE テーブルの合計約 190,000 ローのうち、約 65,000 ローにロー外 LOB カラム DYPNSOURCE があり、そのデータ長は論理ページ・サイズ 16 K 内に余裕で収まっていました。

上記出力で minlen と avglen の間にあったロー内 LOB 長を選択することで、さまざまな個数のロー外 LOB をロー内に持ち込むことができ、結果として LOB 記憶領域の節約量をさまざまに変えることができます。

ロー内 LOB カラムを含むテーブルでのダウングレード

ロー内 LOB カラムで定義されたテーブルがある場合、テーブルのそのカラムに実際にデータが含まれるかどうかにかかわらず、Adaptive Server をダウングレードすることはできません。これは、適用されます。

このような Adaptive Server をダウングレードする必要がある場合、影響を受けるテーブルからデータをいったんすべてコピー (bcp out) してから、元に戻し (bcp in) ます。

第 22 章

instead of トリガの使用

instead of トリガは、トリガ文 (insert、update、および delete) のデフォルトのアクションを上書きして、ユーザ定義のアクションを実行する特殊なストアド・プロシージャです。

instead of トリガは、for トリガと同様に、特定のビューでデータ修正文が実行されるたびに実行します。for トリガはテーブルの insert/update/delete 文の後で起動するので、after トリガと呼ばれることもあります。1 つの instead of トリガは 1 つの特定のトリガ・アクションに適用できます。

```
instead of update
```

また、複数のアクションに適用することもできます。その場合は、同じトリガがリスト上のすべてのアクションを実行します。

```
instead of insert,update,delete
```

for トリガと同様に、instead of トリガは、トリガがアクティブな間に変更されたレコードを論理的な inserted テーブルと deleted テーブルを使用して格納します。これらのテーブルの各カラムは、トリガで参照されるベース・ビューのカラムに直接マップします。たとえば、V1 という名前のビューに C1、C2、C3、C4 のカラムがあると、トリガによってカラム C1 と C3 だけが変更される場合でも、inserted テーブルと deleted テーブルには、4 つのカラムすべての値が含まれます。Adaptive Server は inserted テーブルと deleted テーブルをメモリに存在するオブジェクトとして自動的に作成して管理します。

instead of トリガを使用すると、ビューが更新をサポートできるようになり、バッチの一部を拒否してその他を成功させるコード・ロジックを実装できます。

instead of トリガはデータ修正文 1 つにつき 1 回だけ起動します。while ループを含む複雑なクエリは、update 文または insert 文を何度も繰り返し、そのたびに instead of トリガを起動できます。

inserted 論理テーブルと deleted 論理テーブル

deleted テーブルと inserted テーブルは、論理 (概念) テーブルです。inserted テーブルは insert 文の挿入値のローを格納している疑似テーブルで、deleted テーブルは update 文の更新値 (更新後のイメージ) のローを格納している疑似テーブルです。inserted テーブルと deleted テーブルのスキーマは、instead of トリガが定義されているビュー、つまりユーザ・アクションが試行されるビューのスキーマと同じです。これらのテーブルの違いは、inserted テーブルと deleted テーブルのすべてのカラムは、対応するビューのカラムが null を扱えない場合でも、null を扱える点です。たとえば、ビューにデータ型 char のカラムがあり、insert 文で挿入値 char を指定すると、挿入テーブルの対応するカラムにはデータ型 varchar があり、入力値は varchar に変換されます。ただし、値を挿入テーブルに追加すると、後続のブランクが値から切り捨てられません。

指定した値のデータ型が、挿入先カラムのデータ型と異なる場合は、値が内部でカラムのデータ型に変換されます。変換に成功すると変換後の値がテーブルに挿入されますが、変換に失敗した場合は、文がアボートされます。この例では、ビューがテーブルから整数のカラムを選択する場合、

```
CREATE VIEW v1 AS SELECT intcol FROM t1
```

値 1.0 は整数値 1 に問題なく変換できるため、次の insert 文によって instead of が v1 の実行をトリガします。

```
INSERT INTO v1 VALUES (1.0)
```

一方、次の文は例外を引き起こし、instead of トリガが実行する前にアボートされます。

```
INSERT INTO v1 VALUES (1.1)
```

deleted テーブルと inserted テーブルをトリガで調べ、トリガ・アクションを実行すべきかどうか、またその実行方法を判断できますが、トリガ・アクションでテーブル自体を変更することはできません。

deleted テーブルは delete と update、inserted テーブルは insert と update と一緒に使用されます。

注意 instead of トリガは inserted テーブルと deleted テーブルをメモリ内のテーブルまたはワークテーブルとして作成します。for トリガはトランザクション・ログ syslogs を読み込んで、inserted テーブルと deleted テーブルのローを実行中に生成します。

LOB のデータ型

LOB (ラージ・オブジェクト) データ型の inserted テーブルまたは deleted テーブルのカラムの値はメモリに格納され、これらのテーブル内に多数のローがあり、ローに大きい LOB 値が含まれている場合は、メモリが大量に使用される可能性があります。

トリガおよびトランザクション

rollback trigger コマンドと rollback transaction コマンドは両方とも instead of トリガに使用されます。rollback trigger は、トリガ文で起動したネスト型のすべての instead of トリガで行われた作業をロールバックします。rollback transaction は、全トランザクションで行われた作業を最新の savepoint までロールバックします。

ネストと再帰

for トリガと同様に、instead of トリガは 16 レベルにネストできます。現在のネスト・レベルは、*@@nestlevel* に格納されます。システム管理者は、設定パラメータ allow nested triggers を使用してトリガのネストを on または off にできます。ネストは、デフォルトではオンになっています。トリガのネストを有効にすると、別のトリガを含んでいるテーブルを変更するトリガが 2 番目のトリガを起動し、この 2 番目のトリガが 3 番目のトリガを起動するという無限のループが生じます。この場合は、ネスト・レベルが超過するとプロセスが終了し、トリガはアボートされます。どのネスト・レベルでも、トリガのロールバック・トランザクションは各トリガの結果をロールバックし、トランザクション全体を取り消します。ロールバック・トリガは、ネストされたトリガと、最初のトリガを実行させたデータ修正文にだけ作用します。

instead of トリガと for トリガのネストをインターリーブできます。たとえば、ビューの instead of update トリガを持つ update 文はトリガを実行します。for トリガを定義してテーブルを更新する SQL 文がトリガに含まれている場合は、そのトリガが起動します。for トリガに、instead of トリガを持つ別のビューを更新する SQL 文が含まれ、後でそれが実行されるような場合もあります。

再帰

ただし、instead of トリガと for トリガの再帰の動作は異なります。for トリガは再帰をサポートしていますが、instead of トリガは再帰をサポートしていません。instead of トリガが起動したビューと同じビューを参照している場合、トリガは再帰的に呼び出されず、トリガ文が直接ビューに適用されます。つまり、文はビューの基盤となるベース・テーブルに対する修正として解決されます。この場合、ビューの定義が更新可能なビューのすべての制約を満たしている必要があります。ビューが更新不可の場合は、エラーが発生します。

たとえば、トリガがビューの instead of update トリガとして定義されている場合は、instead of トリガ内の同じビューに update 文を実行しても、トリガが再実行することはありません。トリガで実行される更新は、ビューに instead of トリガがなかった場合と同様に、ビューに対して処理されます。更新によって変更されたカラムは、1 つのベース・テーブルに解決される必要があります。

instead of insert トリガの使用

instead of insert トリガをビューに定義して、insert 文の標準的なアクションを置き換えることができます。通常、このトリガは 1 つまたは複数のベース・テーブルにデータを挿入するためにビューに定義されます。

ビュー select リストのカラムは、null 入力可能または null 入力不可の場合があります。view カラムに null を入力できない場合は、ローをビューに挿入する SQL 文でカラムの値を指定する必要があります。ビューの non-nullable カラムには、null 以外の有効な値に加え、明示的な null の値も受け入れられます。view カラムを定義している式に以下のような項目が含まれている場合は、view カラムに null が許可されます。

- null を許可するベース・テーブル・カラムの参照
- 算術演算子
- 関数の参照
- null を許可するサブ式を持つ CASE
- nullif

sp_help は、ビューのどのカラムが null を許可するかをレポートします。

instead of insert トリガを持つビューを参照している insert 文は、null を許可しないビューのカラムごとに値を指定する必要があります。これには以下のように、入力値を指定できないベース・テーブルのカラムを参照するビューのカラムも含まれます。

- ベース・テーブルの computed カラム
- identity insert が OFF に設定されているベース・テーブルの identity カラム。

挿入テーブルのデータを使用するベース・テーブルに対する instead of insert トリガに insert 文が含まれている場合、insert 文はこのようなカラムの値を文の select リストに含めずに無視する必要があります。ビューに対する insert 文はこれらのカラムにダミー値を生成できますが、instead of insert トリガの insert 文はそれらの値を無視し、Adaptive Server が正しい値を指定します。

例

insert 文は、ベース・テーブルの identity または computed カラムにマップする view カラムの値を指定する必要があります。ただし、プレースホルダの値を指定することもできます。値をベース・テーブルに挿入する instead of トリガの insert 文は、指定した値を無視するように記述されます。

たとえば、これらの文はプロセスを説明するテーブル、ビュー、およびトリガを作成します。

```
CREATE TABLE BaseTable
(PrimaryKey         int IDENTITY
Color               varchar (10) NOT NULL,
Material            varchar (10) NOT NULL,
TranTime            timestamp
)
---------------------
--Create a view that contains all columns from the base table.
CREATE VIEW  InsteadView
AS SELECT PrimaryKey, Color, Material, TranTime
FROM BaseTable
---------------------
Create an INSTEAD OF INSERT trigger on the view.
CREATE TRIGGER InsteadTrigger on InsteadView
INSTEAD OF INSERT
AS
BEGIN
    --Build an INSERT statement ignoring
    --inserted.PrimaryKey and
    --inserted.TranTime.
INSERT INTO BaseTable
    SELECT Color, Material
    FROM inserted
END
```

BaseTable を直接参照する insert 文は、PrimaryKey カラムと TranTime カラムの値を指定できません。次に例を示します。

```
--A correct INSERT statement that skips the PrimaryKey
--and TranTime columns.
INSERT INTO BaseTable (Color, Material)
            VALUES ('Red', 'Cloth')

--View the results of the INSERT statement.
SELECT PrimaryKey, Color, Material, TranTime
FROM BaseTable

--An incorrect statement that tries to supply a value
--for the PrimaryKey and TranTime columns.
INSERT INTO BaseTable
    VALUES (2, 'Green', 'Wood', 0x0102)
```

```
INSERT statements that refer to InsteadView, however,
    must supply a value for PrimaryKey:

--A correct INSERT statement supplying a dummy value for
--the PrimaryKey column.A value for TranTime is not
--required because it is a nullable column.
INSERT INTO InsteadView (PrimaryKey, Color, Material)
        VALUES (999, 'Blue', 'Plastic')
--View the results of the INSERT statement.
SELECT PrimaryKey, Color, Material, TranTime
FROM InsteadView
```

InsteadTrigger に渡された inserted テーブルは、null を扱えない PrimaryKey カラムで構成されています。したがって、ビューを参照している insert 文がこのカラムの値を指定する必要があります。値 999 は InsteadTrigger に渡されますが、InsteadTrigger の insert 文は inserted.PrimaryKey を選択しないので、値は無視されます。BaseTable に挿入されたローは、実際には PrimaryKey に 2、TranTime には Adaptive Server で生成された timestamp 値があります。

default 定義の not null カラムが instead of insert トリガを持つビューで参照されている場合は、ビューを参照している insert 文がカラムの値を指定する必要があります。この値は、トリガに渡される inserted テーブルを構築する必要があります。デフォルト値を使用すべきであることをトリガに知らせる値に規則が必要です。insert 文で not null カラムに明示的な null 値を指定するなどの規則が考えられます。instead of insert トリガは、ビューが定義されているテーブルに挿入すると、明示的な null を無視できるため、テーブルにデフォルト値が挿入されます。次に例を示します。

```
--Create a base table with a not null column that has
--a default
CREATE TABLE td1 (coll int DEFAULT 9 NOT NULL, col2 int)

--Create a view that contains all of the columns of
--the base table.
CREATE VIEW vtd1 as select * from td1
--create an instead of trigger on the view
CREATE TRIGGER vtd1insert on vtd1 INSTEAD OF INSERT AS
BEGIN
    --Build an INSERT statement that inserts all rows
    --from the inserted table that have a NOT NULL value
    --for col1.
    INSERT INTO td1 (col1,col2) SELECT * FROM inserted
    WHERE col != null

    --Build an INSERT statement that inserts just the
    --value of col2 from inserted for those rows that
    --have NULL as the value for col1 in inserted.In
    --this case, the default value of col1 will be
    --inserted.
    INSERT INTO td1 (col2) SELECT col2 FROM inserted
    WHERE col1 = null
```

```
END
```

instead of insert トリガの deleted テーブルは常に空です。

instead of update トリガ

通常、instead of update トリガは 1 つまたは複数のベース・テーブルのデータを変更するためにビューに定義されます。

instead of update トリガを持つビューを参照する update 文では、サブセット内のカラムが null 入力不可能なカラムかどうかにかかわらず、update 文の set 句に出現する場合があります。

更新できないビュー・カラム (入力値を指定できないベース・テーブルのカラムを参照しているカラム) でも更新文の set 句に出現する場合があります。更新できないカラムには、次のものがあります。

- ベース・テーブルの computed カラム
- identity insert が off に設定されているベース・テーブルの identity カラム
- timestamp データ型のベース・テーブル・カラム。

通常、テーブルを参照している update 文で、computed カラム、identity カラム、または timestamp カラムの値を設定しようとすると、エラーが生成されます。これは、これらのカラムの値は Adaptive Server が生成する必要があるからです。ただし、update 文が instead of update トリガを持つビューを参照する場合は、トリガに定義されているロジックがこれらのカラムをバイパスしてエラーを回避します。そのため、instead of update トリガは、ベース・テーブルの対応するカラムの値を更新できません。これを行うには、update 文の set 句のカラムをトリガの定義から除外します。

この解決方法が良い理由は、更新不可の挿入カラムのデータを instead of update トリガが処理する必要がないからです。挿入テーブルには、set 句で指定されていないカラムの値が含まれています。これらは update 文が発行される前に存在していたものです。トリガは if update (カラム) 句を使用して、特定のカラムが更新されたかどうかをテストできます。

instead of update トリガは、computed カラム、identity カラム、または timestamp カラムに指定された値を検索条件 where 句でのみ使用する必要があります。ビューの instead of update トリガが、computed カラム、identity カラム、timestamp カラム、またはデフォルト・カラムの更新値を処理するために使用するロジックは、これらのカラム・タイプの挿入値に適用されるロジックと同じです。

inserted テーブルと deleted テーブルにはそれぞれ、更新の条件を満たすすべてのローに対して、instead of update トリガを持つビューの update 文にローが含まれています。inserted テーブルのローには、更新オペレーションの後のカラムの値が含まれ、deleted テーブルのローには、更新オペレーションの前のカラムの値が含まれています。

instead of delete トリガ

instead of delete トリガは、delete 文の標準的なアクションを置き換えることができます。通常、instead of delete トリガはベース・テーブルのデータを変更するためにビューに定義されます。

delete 文は既存のデータ値の変更は指定せず、削除するローのみを指定します。delete テーブルは、delete 文の削除された値を持つロー、または update 文の更新前の値 (更新前のイメージ) を格納している疑似テーブルです。instead of delete トリガに送られた deleted テーブルには、delete 文が発行される前に存在したローのイメージが含まれています。deleted テーブルの形式は、ビューに定義されている select リストの形式に基づきますが、例外として、not null カラムはすべて null 入力可能カラムに変換されます。

delete トリガに渡された inserted テーブルは常に空です。

searched および positioned *update* と *delete*

update 文と delete 文を検索または配置することができます。searched delete は、where 句にオプションの述語式を含んでおり、この句が削除するローを修飾します。searched update は、where 句にオプションの述語式を含んでおり、この句が更新するローを修飾します。

次に、searched delete 文の例を示します。

```
DELETE myview WHERE myview.col1 > 5
```

この文は、myview のすべてのローを調べ、where 句で指定された述部 (myview.col1 > 5) を適用して実行され、どのローを削除すべきかを決定します。

ジョインは、searched update 文と delete 文では許可されていません。別のテーブルのローを使用してビューの条件を満たすローを見つけるには、サブクエリを使用します。たとえば、この文は許可されません。

```
DELETE myview FROM myview, mytab
    where myview.col1 = mytab.col1
```

一方、これと等価なサブクエリを使った文は許可されます。

```
DELETE myview WHERE col1 in (SELECT col1 FROM mytab)
```

positioned update 文と delete 文は、カーソルの結果セットにのみ実行され、1 つのローのみに影響を与えます。次に例を示します。

```
DECLARE mycursor CURSOR FOR SELECT * FROM myview
OPEN mycursor
FETCH mycursor
DELETE myview WHERE CURRENT OF mycursor
```

positioned delete 文は、mycursor が現在置かれている myview のローのみを削除します。

instead of トリガがビューに存在する場合は常に、条件を満たす searched delete または update 文、つまりジョインのない文に実行されます。instead of トリガが positioned delete または update 文に実行されるためには、次の 2 つの条件を満たす必要があります。

- カーソルが宣言されるとき、つまりコマンド declare cursor が実行されるときに、instead of トリガが存在します。
- カーソルを定義する select 文はビューのみにアクセスできます。たとえば、select 文にはジョインが含まれていませんが、ビューのカラムのどのサブセットにでもアクセスできます。

instead of トリガは、positioned delete または update 文がスクロール可能なカーソルに実行されるときに実行します。ただし、**client cursor** とコマンド set cursor rows を使用している場合は、instead of トリガが起動しません。

クライアント・カーソル

カーソルを処理するクライアント・カーソルは、Open Client ライブラリ関数を使用してアプリケーションで宣言され、フェッチされます。Open Client ライブラリ関数は fetch コマンド 1 つで Adaptive Server から複数のローを取得して、これらのローをバッファに入れ、後続の fetch コマンドで一度に 1 つずつローをアプリケーションに返します。バッファに入れたローがすべて読み取られるまで、Adaptive Server からこれ以上のローを取得する必要はありません。デフォルトでは、fetch コマンドを 1 回受け取るごとに Adaptive Server は 1 つのローを Open Client ライブラリ関数に返します。ただし、コマンド set cursor rows を使用すると、Adaptive Server が返すローの数を変更できます。

fetch のたびに返されるローの数を増やすために set cursor rows が使用されないクライアント・カーソルの Positioned update および delete 文によって、instead of トリガが実行します。ただし、set cursor rows が fetch コマンドのたびに返されるローの数を増やす場合、instead of トリガは declare cursor の内部処理中にカーソルが読み込み専用とマークされていない場合にのみ実行します。次に例を示します。

```
--Create a view that is read-only (without an instead
--of trigger) because it uses DISTINCT.
CREATE VIEW myview AS
SELECT DISTINCT (col1) FROM tab1
```

```
--Create an INSTEAD OF DELETE trigger on the view.
CREATE TRIGGER mydeltrig ON myview
INSTEAD OF DELETE
AS
BEGIN
    DELETE tab1 WHERE col1 in (SELECT col1 FROM deleted)
END

Declare a cursor to read the rows of the view
DECLARE cursor1 CURSOR FOR SELECT * FROM myview

OPEN cursor1

FETCH cursor1

--The following positioned DELETE statement will
--cause the INSTEAD OF TRIGGER, mydeltrig, to fire.
DELETE myview WHERE CURRENT OF cursor1

--Change the number of rows returned by ASE for
--each FETCH.
SET CURSOR ROWS 10 FOR cursor1

FETCH cursor1

--The following positioned DELETE will generate an
--exception with error 7732: "The UPDATE/DELETE WHERE
--CURRENT OF failed for cursor 'cursor1' because
--the cursor is read-only."
DELETE myview WHERE CURRENT OF cursor1
```

set cursor rows を使用すると、Adaptive Server とアプリケーションのカーソル位置の間にずれが生じます。Adaptive Server は、返された 10 個のローの最後のローに配置されますが、アプリケーションは 10 個のうちどのローにでも配置できます。これは、Open Client ライブラリ関数が Adapter Server に情報を送信せずにローをバッファに入れてスクロールするからです。アプリケーションによって positioned delete 文が送信されると、Adaptive Server は cursor1 の位置を判別できないため、Open Client ライブラリ関数も cursor1 がアプリケーションに置かれているローのカラムのサブセットの値を送信します。これらの値は、positioned delete を searched delete 文に変換するために使用されます。つまり、col1 の値が cursor1 の現在のローで 5 の場合、Adaptive Server は `'where col1 = 5'` のような句を使用してローを見つけます。

positioned delete が searched delete に変換されるとき、instead of トリガのないテーブルやビューのカーソルと同様に、カーソルが更新可能でなければなりません。上の例では、cursor1 を定義している select 文が、myview を定義している select 文で置き換えられます。

```
DECLARE cursor1 CURSOR FOR SELECT * FROM myview
```

変換後は、次のようになります。

```
DECLARE cursor1 CURSOR FOR SELECT DISTINCT (col1)
FROM tab1
```

select リストに distinct オプションがあるため、cursor1 は更新不可、つまり読み込み専用です。これが原因で、positioned delete が処理されるときに 7732 エラーが発生します。

select 文を定義してビューを置き換えた結果のカーソルが更新可能な場合は、set cursor rows を使用するかどうかにかかわらず instead of トリガが起動します。Adaptive Server がサポートしている他の種類のカーソルでは、set cursor rows を使用しても、instead of トリガの起動を阻止することはできません。

トリガに関する情報の取得

instead of トリガに関する情報は for トリガの情報として格納されます。

- トリガの定義クエリ・ツリーは sysprocedures に格納されます。
- トリガにはそれぞれ識別番号 (オブジェクト ID) があり、sysobjects の新しいローに格納されます。トリガが適用されるビューのオブジェクト ID は、トリガの sysobjects ローの deltrig、instrig、および updtrig カラムに格納されます。トリガのオブジェクト ID は、トリガを適用するビューの sysobjects ローの deltrig、instrig、または putrid カラムに格納されます。
- syscomments に格納されているトリガのテキストを表示するには、sp_helptext を使用します。システム・セキュリティ担当者が sp_configure を使用してカラム・パラメータ allow select on syscomments.text をリセットした場合、sp_helptext を使用してトリガのテキストを表示するには、トリガの作成者かシステム管理者でなければなりません。
- トリガに関するレポートを取得するには、sp_help を使用します。sp_help は instead of トリガを instead of トリガの *object_type* としてレポートします。
- instead of トリガによって参照されるビューについてレポートするには、sp_depends を使用します。sp_depends *view_name* はトリガ名とそのタイプを instead of トリガとしてレポートします。

第 23 章

トランザクション：データの一貫性およびリカバリ

「トランザクション」は、一連の Transact-SQL 文を、1 つの単位として処理します。グループ内の文は、すべて実行されるか、まったく実行されないかのどちらかになります。

トランザクションの機能

Adaptive Server は、すべてのデータ修正コマンド (1 つの手順で行う変更要求を含む) を、トランザクションとして自動的に管理します。デフォルトでは、insert、update、delete 文のそれぞれが 1 つのトランザクションとみなされます。

ただし、次のような場合には注意が必要です。ユーザの Lee が、authors、titles、titleauthors の各テーブルに対して、一連のデータ検索と修正を行うとします。Lee がこの作業を実行しているときに、別のユーザ Lil が titles テーブルの更新作業を始めました。この場合、Lil がテーブルを更新すると、Lee が行っている作業の結果との整合性が失われるおそれがあります。これを避けるには、Lee の一連の文を 1 つのトランザクションにグループ化します。これによって、テーブル内での Lee が作業中の部分はロックされ、Lil が行った更新は反映されません。これで Lee は正確なデータに基づいて必要な作業を完了できます。Lee がテーブルの更新を終了した後、Lil が行った更新がテーブルに反映されます。

トランザクションを作成するには、次のコマンドを使用します。

- begin transaction – トランザクション・ブロックの開始点を表すコマンド。構文は次のとおりです。

```
begin {transaction | tran} [transaction_name]
```

transaction_name は、トランザクションに割り当てる名前です。トランザクション名は識別子の規則に従います。トランザクション名を指定できるのは、ネストされた begin/commit または begin/rollback 文の最も外側にある文のペアに対してだけです。

- save transaction － トランザクション内のセーブポイントを表すコマンド。

  ```
  save {transaction | tran} savepoint_name
  ```

  savepoint_name は、セーブポイントに割り当てる名前です。識別子の規則に従います。

- commit － トランザクション全体をコミットする。

  ```
  commit [transaction | tran | work]
  [transaction_name]
  ```

- rollback － トランザクションをセーブポイントまで、またはトランザクションの開始点までロールバックする。

  ```
  rollback [transaction | tran | work]
  [transaction_name | savepoint_name]
  ```

たとえば、ユーザの Lee が、『The Gourmet Microwave』という本の作家 2 人に支払う印税の分配率を更新するとします。2 人の作家についてのデータを別々に変更すると、データベースに整合性がなくなってしまいます。そこで、次に示すように 2 つの文を 1 つのトランザクションとしてグループ化します。

```
begin transaction royalty_change

update titleauthor
set royaltyper = 65
from titleauthor, titles
where royaltyper = 75
and titleauthor.title_id = titles.title_id
and title = "The Gourmet Microwave"

update titleauthor
set royaltyper = 35
from titleauthor, titles
where royaltyper = 25
and titleauthor.title_id = titles.title_id
and title = "The Gourmet Microwave"

save transaction percentchanged

/* After updating the royaltyper entries for
** the two authors, insert the savepoint
** percentchanged, then determine how a 10%
** increase in the book's price would affect
** the authors' royalty earnings.*/

update titles
```

```
set price = price * 1.1
where title = "The Gourmet Microwave"

select (price * total_sales) * royaltyper
from titles, titleauthor
where title = "The Gourmet Microwave"
and titles.title_id = titleauthor.title_id

/* The transaction is rolled back to the savepoint
** with the rollback transaction command.*/

rollback transaction percentchanged

commit transaction
```

トランザクションを使用することによって、次のことが保証されます。

- 一貫性 – 同時に行われるクエリや変更の要求が衝突することなく、ユーザが変更途中のデータを操作したり見たりすることはできない。
- リカバリ – システムに障害が発生したとき、データベースの回復が完全かつ自動的に行われる。

トランザクションが SQL 規格に準拠するように、Adaptive Server では、トランザクションのモードおよび独立性レベルを選択できます。SQL 規格に準拠するトランザクションを必要とするアプリケーションでは、すべてのセッションの最初にこれらのオプションを設定する必要があります。「トランザクション・モードおよび独立性レベルの選択」(670 ページ) を参照してください。

トランザクションと一貫性

マルチユーザ環境の場合、同時に行われるクエリやデータ変更要求が、互いに干渉するのを防止する必要があります。あるクエリの実行中にその処理の対象となるデータが別のユーザの更新によって変更されてしまうと、クエリの結果に一貫性がなくなってしまいます。

Adaptive Server は、各トランザクションに対して適切なのロックのレベルを自動的に設定します。select 文に holdlock キーワードを指定すると、共有ロックをより厳密にクエリごとに制限できます。

トランザクションとリカバリ

トランザクションは処理の単位であり、リカバリの単位でもあります。Adaptive Server は、1 段階の手順で行う変更要求をトランザクションとして処理するため、システムに障害が起こった場合には、データベースの完全なリカバリが可能になります。

Adaptive Server のリカバリ時間は、秒単位と分単位で測定されます。ユーザはリカバリ時間の最大値を指定できます。

リカバリとバックアップに関連する SQL コマンドについては、「トランザクションのバックアップとリカバリ」(691 ページ) を参照してください。

注意 多数の Transact-SQL コマンドをグループ化し、実行に長時間かかるトランザクションにすると、リカバリ時間に影響することがあります。トランザクションがコミットされる前に Adaptive Server に障害が発生すると、リカバリ時間は長くなります。これは、Adaptive Server がそのトランザクションを取り消す必要があるためです。

コンポーネント統合サービスを有効にした状態でリモート・データベースを使用している場合、トランザクションはいくつかの異なる方法で処理されます。コンポーネント統合サービス・ユーザーズ・ガイドを参照してください。

Adaptive Server の DTM 機能を購入し、インストールしている場合、複数のサーバにあるデータを更新するトランザクションにおいても、トランザクションの一貫性は重要な役割を果たします。『Adaptive Server 分散トランザクション管理機能の使用』を参照してください。

トランザクションの使用

begin transaction および commit transaction コマンドは、Adaptive Server に、複数のコマンドを 1 つにまとめて処理するように指示します。rollback transaction コマンドは、処理の開始時または「セーブポイント」まで、トランザクションを戻します。save transaction コマンドを使用して、トランザクション内にセーブポイントを設定できます。

複数の SQL 文をグループ化して 1 つの単位として動作させること以外に、システムのパフォーマンスを向上させることもできます。これは、システムの 1 回のオーバヘッドがコマンドごとでなくトランザクションごとに発生するためです。

トランザクションは、すべてのユーザが定義できます。トランザクション・コマンドを使用するために必要なパーミッションはありません。

トランザクションでのデータ定義コマンドの使用

ddl in tran データベース・オプションを true に設定すると、create table、grant、alter table などの特定のデータ定義言語コマンドをトランザクションで使用できます。model データベースで ddl in tran オプションが true に設定されている場合は、その model データベースで ddl in tran オプションが true に設定された後に作成されたすべてのデータベースのトランザクション内で、上記のコマンドを実行できます。ddl in tran の現在の設定値を確認するには、sp_helpdb を使用します。

警告！ データ定義コマンドを使用するときには注意してください。トランザクション内でデータ定義言語コマンドを使用する状況でのみ、create schema 文を使って行います。データ定義言語コマンドは、sysobjects テーブルなどのシステム・テーブル上にロックを設定します。データ定義言語コマンドをトランザクション内で実行する場合には、トランザクションを短くしてください。

トランザクション内の tempdb に対しては、データ定義言語コマンドを使用しないでください。これは、パフォーマンスを低下させます。tempdb には、常に ddl in tran オプションを false に設定してください。

---

ddl in tran を true に設定するには、次のように入力します。

```
sp_dboption database_name,"ddl in tran", true
```

次に、そのデータベース内で checkpoint コマンドを実行します。

最初のパラメータは、オプションを設定するデータベース名を指定します。sp_dboption を実行するには、master データベースを使用する必要があります。どのユーザも、パラメータを使用しないで sp_dboption オプションを実行して、現在のオプション設定を表示できます。しかし、オプションを設定できるのは、システム管理者かデータベースの所有者だけです。

sp_dboption の ddl in tran オプションを true に設定している場合にだけ、次に示すコマンドをトランザクション内で使用できます。

- create default
- create index
- create procedure
- create rule
- create schema
- create table
- create trigger
- create view
- drop default

- drop index
- drop procedure
- drop rule
- drop table
- drop trigger
- drop view
- grant
- revoke

master データベースの変更またはテンポラリ・テーブルの作成に関するシステム・プロシージャは、トランザクション内で使用できません。

次のコマンドを、トランザクション内で実行しないでください。

- alter database
- alter table...partition
- alter table...unpartition
- create database
- disk init
- dump database
- dump transaction
- drop database
- load transaction
- load database
- reconfigure
- select into
- update statistics
- truncate table

トランザクション内で使用できないシステム・プロシージャ

次のシステム・プロシージャは、トランザクション内で使用できません。

- sp_helpdb、sp_helpdevice、sp_helpindex、sp_helpjoins、sp_helpserver、sp_lookup、sp_spaceused (これらのシステム・プロシージャは、テンポラリ・テーブルを作成するためです)
- sp_configure
- master データベースを変更するシステム・プロシージャ

トランザクションの開始とコミット

begin transaction と commit transaction コマンドの間に、いくつでも SQL 文とストアド・プロシージャを記述することができます。それぞれの文の構文は、次のとおりです。

```
begin {transaction | tran} [transaction_name]
commit {transaction | tran | work} [transaction_name]
```

transaction_name は、トランザクションに割り当てる名前です。トランザクション名は識別子の規則に従います。

transaction、tran、(commit transaction の) work の各キーワードは同義語です。それぞれのキーワードは同じ意味で使用できます。ただし、transaction と tran は Transact-SQL の拡張機能です。work だけが SQL 標準に準拠しています。

次に例を示します。

```
begin tran
  statement
  procedure
  statement
commit tran
```

トランザクションが実行中でない場合、commit transaction コマンドは Adaptive Server に影響しません。

トランザクションのロールバックと保存

何らかの障害が発生したり、ユーザによって変更が行われるため、トランザクションがコミットする前に取り消される場合は、実行を終了した文やプロシージャをすべて取り消さなければなりません。処理中のロールバックの影響については、表 23-2 (683 ページ) を参照してください。

commit transaction コマンドの実行前であればいつでも、rollback transaction コマンドでトランザクションの取り消し、つまりロールバックができます。セーブポイントを使用すると、トランザクションの全体またはトランザクションを部分的に取り消すことができます。ただし、トランザクションをコミットした後に、取り消すことはできません。

rollback transaction の構文は次のとおりです。

```
rollback {transaction | tran | work}
    [transaction_name | savepoint_name]
```

「セーブポイント」とは、トランザクション内にユーザが設定するマーカで、そのトランザクションが取り消された場合にロールバックをどこまで行うかを示す点です。バッチの特定の部分だけをコミットするには、バッチ全体をコミットする前に、不要な部分をセーブポイントまでロールバックします。

save transaction コマンドを使用して、トランザクション内にセーブポイントを設定します。

```
save {transaction | tran} savepoint_name
```

セーブポイント名は、識別子の規則に従う必要があります。

rollback transaction に *savepoint_name* または *transaction_name* が指定されていない場合、そのトランザクションはバッチ内の最初の begin transaction までロールバックされます。

次に save transaction と rollback transaction コマンドの使用方法を示します。

```
begin tran
    statements                  グループ A
    save tran mytran
    statements                  グループ B
    rollback tran mytran        グループ B をロールバック
    statements                  グループ C
    commit tran                 グループ A とグループ C をコミット
```

commit transaction コマンドが発行されるまで、Adaptive Server は、別の begin transaction を検出しないかぎり、後続の文をすべてそのトランザクションの一部であるとみなします。その時点では、Adaptive Server は後続の文すべてをネストされた新しいトランザクションの一部であるとみなします。「ネストされたトランザクション」(669 ページ) を参照してください。

トランザクションが実行中でない場合は、rollback transaction または save transaction コマンドは Adaptive Server に対して影響を与えず、エラー・メッセージが表示されることもありません。

save transaction コマンドを使用すると、バッチや他のプロシージャに影響を与えずにトランザクションをロールバックできるように、ストアド・プロシージャまたはトリガ内でトランザクションを作成できます。次に例を示します。

```
create proc myproc as
  begin tran
save tran mytran
statements
if ...
    begin
        rollback tran mytran
    /*
```

```
        ** Rolls back to savepoint.
        */
          commit tran
        /*
        ** This commit needed; rollback to a savepoint
        ** does not cancel a transaction.
        */
       end
    else
    commit tran
        /*
        ** Matches begin tran; either commits
        ** transaction (if not nested) or
        ** decrements nesting level.
        */
```

セーブポイントまでロールバックしない場合、トランザクション名を指定できるのは、begin/commit または begin/rollback の最も外側にある文のペアに対してだけです。

警告！ ネストされたトランザクション文でトランザクション名を使用すると、そのトランザクション名は無視されるか、エラーの原因になる場合があります。使用するトランザクションが存在しているストアド・プロシージャまたはトリガが、他のトランザクション内から呼び出される可能性がある場合は、トランザクション名を指定しないでください。

トランザクションのステータスの確認

グローバル変数 *@@transtate* では、トランザクションの現在のステータスを追跡できます。Adaptive Server は、ある文を実行した後のトランザクションの変更を追跡し、どのようなステータスを返すか決定します。

表 23-1: @@transtate 値

値	意味
0	トランザクションが進行中。トランザクションは有効であり、前に実行した文は正常に終了した。
1	トランザクションが成功した。トランザクションが完了し、その変更がコミットされた。
2	文がアボートされた。前の文がアボートされた。トランザクションには影響しない。
3	トランザクションがアボートされた。トランザクションがアボートされ、すべての変更がロールバックされた。

各文の実行が終了しても、グローバル変数 *@@transtate* の値は Adaptive Server によってクリアされません。トランザクション内で、文 (たとえば insert 文など) の後に *@@transtate* を記述すると、その文が正常に実行されたか、アボートされたかを判断し、トランザクションへの文の効果を判断できます。次の例では、トランザクション中 (正常終了した insert の後) と、トランザクションがコミットされた後で、*@@transtate* の値をチェックしています。

```
begin transaction
insert into publishers (pub_id) values ("9999")

(1 row affected)

select @@transtate

----------
         0

(1 row affected)

commit transaction
select @@transtate

----------
         1

(1 row affected)
```

次の例では、ルール違反のため失敗した insert 文の後、およびトランザクションがロールバックされた後で、*@@transtate* の値をチェックしています。

```
begin transaction
insert into publishers (pub_id) values ("7777")

Msg 552, Level 16, State 1:
A column insert or update conflicts with a rule bound to the
column.The command is aborted.The conflict occured in database
'pubs2', table 'publishers', rule 'pub_idrule', column
'pub_id'.

select @@transtate

----------
        2

(1 row affected)

rollback transaction
select @@transtate

----------
        3

(1 row affected)
```

@@transtate の値は、トランザクションが行ったアクションに応答することによってのみ変更されます。構文エラーとコンパイル・エラーは、*@@transtate* の値に影響しません。

ネストされたトランザクション

トランザクション内に、別のトランザクションをネストできます。begin transaction と commit transaction 文をネストする場合には、最も外側のペアが実際にトランザクションを開始し、コミットします。内部のペアは、ネストのレベルを追跡するだけです。Adaptive Server は、最も外側の begin transaction に一致する commit transaction が発行されるまで、トランザクションをコミットしません。通常、このようなトランザクションの「ネスト」が起こるのは、begin/commit 文の対を持つストアド・プロシージャまたはトリガが呼び出された場合です。

@@trancount グローバル変数は、トランザクションの現在のネスト・レベルを表します。最初の明示的または暗黙的な begin transaction 文は *@@trancount* の値を 1 に設定します。*@@trancount* の値は、後続の begin transaction によって 1 つずつ増加し、commit transaction 文によって 1 つずつ減少します。*@@trancount* はトリガの起動によっても 1 つずつ増加し、トランザクションはこのトリガの起動文によって開始されます。ネストされたトランザクションは、*@@trancount* が 0 になるまでコミットされません。

たとえば、次のようなネストされた文のグループは、最後の commit transaction が実行されるまで Adaptive Server によってコミットされません。

```
begin tran
    select @@trancount
    /* @@trancount = 1 */
      begin tran
      select @@trancount
      /* @@trancount = 2 */
        begin tran
            select @@trancount
            /* @@trancount = 3 */
      commit tran
    commit tran
commit tran
select @@trancount
/* @@trancount = 0 */
```

トランザクション名またはセーブポイント名を指定しないで rollback transaction 文をネストすると、常に最も外側の begin transaction 文までロールバックされ、トランザクションが取り消されます。

トランザクションの例

トランザクションを指定する方法を、次の例で示します。

```
begin transaction royalty_change
/* A user sets out to change the royalty split */
/* for the two authors of The Gourmet Microwave.*/
/* Since the database would be inconsistent */
/* between the two updates, they must be grouped */
```

```
/* into a transaction.*/
update titleauthor
set royaltyper = 65
from titleauthor, titles
where royaltyper = 75
and titleauthor.title_id = titles.title_id
and title = "The Gourmet Microwave"
update titleauthor
set royaltyper = 35
from titleauthor, titles
where royaltyper = 25
and titleauthor.title_id = titles.title_id
and title = "The Gourmet Microwave"
save transaction percent_changed
/* After updating the royaltyper entries for */
/* the two authors, the user inserts the */
/* savepoint "percent_changed," and then checks */
/* to see how a 10 percent increase in the */
/* price would affect the authors' royalty */
/* earnings.*/
update titles
set price = price * 1.1
where title = "The Gourmet Microwave"
select (price * royalty * total_sales) * royaltyper
from titles, titleauthor, roysched
where title = "The Gourmet Microwave"
and titles.title_id = titleauthor.title_id
and titles.title_id = roysched.title_id
rollback transaction percent_changed
/* The transaction rolls back to the savepoint */
/* with the rollback transaction command.*/
/* Without a savepoint, it would roll back to */
/* the begin transaction.*/
commit transaction
```

トランザクション・モードおよび独立性レベルの選択

Adaptive Server には、SQL 標準に準拠する必要のあるトランザクションをサポートするためのオプションがあります。

- 「トランザクション・モード」オプションには、トランザクションを開始するために暗黙的な begin transaction 文を使用するか使用しないかを設定する。
- 「独立性レベル」オプションには、トランザクションの実行中に他のユーザがデータにアクセスできるレベルを指定する。

これらのオプションの設定は、SQL 規格に準拠したトランザクションを必要とする各セッションの開始時に行ってください。

トランザクション・モードの選択

Adaptive Server は、次のトランザクション・モードをサポートしています。

- 「連鎖モード」では、すべてのデータ検索文およびデータ修正文 (delete、insert、open、fetch、select、および update) の前には暗黙的にトランザクションを開始します。ただし、commit transaction または rollback transaction を使ってトランザクションを明示的に終了してください。
- デフォルトの「非連鎖」トランザクション・モード。Transact-SQL モードとも呼ばれます。このモードでトランザクションを終了するには、明示的な begin transaction 文と commit transaction または rollback transaction 文をペアで記述する必要があります。

これらのトランザクション・モードはいずれも set コマンドの chained オプションを使って指定します。ただし、2 つのトランザクション・モードをアプリケーション内で混在させてはいけません。ストアド・プロシージャとトリガの動作はトランザクション・モードによって異なるので、あるモードで作成したプロシージャを別のモードで実行するには特別な処理が必要となる場合があるためです。

SQL 規格では、連鎖モードを使用して、どの SQL データ検索文や修正文も、1 つのトランザクション内で実行することが要求されます。トランザクションは、セッションが開始した後、または前のトランザクションがコミットかアボートした後の最初のデータ検索文や修正文を、自動的に開始します。これが連鎖トランザクション・モードです。

set 文の chained オプションを設定して、現在のセッションをこのモードに設定できます。

ただし、トランザクション内では set chained コマンドを実行できません。非連鎖トランザクション・モードに戻るには、chained オプションを off に設定します。

次の文のグループの結果は、どちらのモードを使用するかによって異なります。

```
insert into publishers
    values ("9906", null, null, null)
begin transaction
delete from publishers where pub_id = "9906"
rollback transaction
```

非連鎖トランザクション・モードの場合、rollback 文は delete 文だけに影響し、publishers に挿入されたローは残っています。一方、連鎖モードの場合は、insert 文によって暗黙的にトランザクションが開始されているため、insert 文を含むこのトランザクションのすべての処理がロールバックされます。

アプリケーション・プログラムおよびアドホック・ユーザ・クエリでは、正しいトランザクション・モードに対応していることが前提になります。どちらのトランザクション・モードを指定するかは、特定のクエリまたはアプリケーションで SQL 規格に準拠したトランザクションを使う必要があるかどうかによって決めます。連鎖トランザクションを使用するアプリケーション (たとえば Embedded SQL プリコンパイラ) では、各セッションを開始するときに連鎖モードを指定します。

トランザクション・モードとネストされたトランザクション

連鎖モードでは、データ検索文や修正文によって暗黙的にトランザクションが開始されますが、begin transaction 文を明示的に記述するだけでトランザクションをネストできます。最初のトランザクションが暗黙的に開始されると、コミットまたはアボートするまで、以降のデータ検索文や修正文によって新しくトランザクションが開始されることはありません。たとえば、次のクエリを実行すると、最初の commit transaction は連鎖モードですべての変更をコミットし、2 番目のコミットは必要ありません。

```
insert into publishers
    values ("9907", null, null, null)
    insert into publishers
        values ("9908", null, null, null)
    commit transaction
commit transaction
```

注意 連鎖モードでは、正常に実行されるかどうかに関係なく、データ検索文や修正文によってトランザクションが開始されます。テーブルにアクセスしない select でもトランザクションは開始されます。

現在のトランザクション・モードのステータスの確認

Adaptive Server の現在のトランザクション・モードは、グローバル変数 *@@tranchained* の値によって確認できます。select *@@tranchained* は、非連鎖モードでは 0、連鎖モードでは 1 を返します。

独立性レベルの選択

ANSI SQL 規格では、トランザクションの独立性レベルを 4 つ定義しています。それぞれの独立性レベルでは、トランザクションを同時に実行している間は許可されない動作の種類を規定しています。上位レベルには、下位レベルで課した制限が含まれます。

- レベル 0 － あるトランザクションで書き込まれたデータが実際のデータであることを保証します。コミットされていないトランザクションが (insert、delete、update 文などを使って) 修正したデータを、他のトランザクションが変更できないようにします。トランザクションがコミットされるまで、他のトランザクションはデータを修正できません。ただし、他のトランザクションはコミットされていないデータを読み込むことができます。その結果、「ダーティ・リード」となります。
- レベル 1 － ダーティ・リードを防止します。あるトランザクションがローを修正し、その変更をコミットする前に別のトランザクションがそのローを読み込むと、ダーティ・リードが発生します。最初のトランザクションで変更箇所までロールバックすると、2 番目のトランザクションによる読み込みは無効になります。レベル 1 は Adaptive Server がサポートするデフォルトの独立性レベルです。
- レベル 2 － 「繰り返し不可能 読み出し」を防止します。このような読み出しは、あるトランザクションがローを読み込み、2 番目のトランザクションがこのローを修正する場合に発生します。2 番目のトランザクションがこの修正内容をコミットすると、これ以降に最初のトランザクションは元の読み込みとは異なる結果を読み込みます。

  Adaptive Server はレベル 2 をデータオンリーロック・テーブルでサポートします。全ページロック・テーブルではサポートしていません。
- レベル 3 － あるトランザクションが読み込んだデータは、そのトランザクションが終了するまで有効であることを保証します。このため、「幻ロー」を防止できます。Adaptive Server は、select 文の holdlock キーワードを使用して、特定のデータに対して読み込みロックを適用することにより、このレベルをサポートします。1 番目のトランザクションが探索条件を満たすローの集合を読み込んだ後、2 番目のトランザクションがそのデータを、insert、delete、update などを使って修正すると、幻ローが発生します。最初のトランザクションが同じ探索条件で読み込みを繰り返すと、別のロー・セットを得ることになります。

セッションに独立性レベルを設定するには、set コマンドの transaction isolation level オプションを使用します。select 文の at isolation 句を使用する場合とは異なり、特定のクエリに対してだけ独立性レベルを適用できます。次に例を示します。

```
set transaction isolation level 0
```

Adaptive Server と ANSI SQL の独立性レベルのデフォルト

デフォルトでは、Adaptive Server のトランザクション独立性レベルは 1 です。ANSI SQL 規格では、すべてのトランザクションのデフォルトの独立性はレベル 3 でなければなりません。このレベルでは、ダーティ・リード、繰り返し不可能読み出し、幻ローが防止されます。この独立性のデフォルト・レベルを設定するために、Transact-SQL は set 文の transaction isolation level 3 オプションを提供しています。このオプションは、トランザクション内のすべての select オペレーションに holdlock を自動的に適用するように、Adaptive Server に指示します。次に例を示します。

```
set transaction isolation level 3
```

transaction isolation level 3 を使用するアプリケーションは、各セッションの始めでその独立性レベルを設定するようにしてください。ただし、transaction isolation level 3 を設定すると、トランザクションの間、Adaptive Server はすべての読み込みロックを保持します。また、連鎖トランザクション・モードを使用する場合、この独立性レベルは、トランザクションを暗黙的に開始するすべてのデータ検索文や修正文に影響します。どちらの場合でも、一部のアプリケーションで同時実行性の問題が発生することがあります。これは、より多くのロックが、より長い間保持される可能性があるからです。

セッションを Adaptive Server のデフォルトの独立性レベルに戻すには、次の手順に従います。

```
set transaction isolation level 1
```

ダーティ・リード

ダーティ・リードの影響を受けないアプリケーションでは、各セッションの始めで transaction isolation level 0 を設定すると、同じデータにアクセスしたとき、同時実行性が向上し、デッドロックが減少します。たとえば、テーブルに保管されているすべての預金口座の瞬間的な平均残高を求めるアプリケーションなどです。アクティブ・テーブルでは残高が頻繁に変わります。しかし、現在の平均残高のスナップショットだけが必要なので、このアプリケーションでは独立性レベル 0 を使用してテーブルを問い合わせます。テーブル内の特定の口座に対する預け入れと引き出しのようなデータの一貫性が必要な他のアプリケーションでは、独立性レベル 0 の使用は避けてください。

独立性レベル 0 で行うスキャンでは、読み込みロックが発生しません。したがってこれらのクエリは、他のトランザクションが同じデータに書き込むことを防ぎません。この逆の場合も同様のことが言えます。ただし、独立性レベルを 0 に設定した場合でも、ユーティリティ (dbcc など) とデータ修正文 (update など) では、スキャンのための読み込みロックが発生します。これは、正しいデータを読み込んでからそのデータを修正したことを保証して、データベースの整合性を維持する必要があるからです。

独立性レベル 0 のスキャンでは読み込みロックが発生しないので、スキャンの実行中にそのスキャンの結果セットが変化する場合があります。基本となるテーブルのデータが変更されたためにスキャン位置が不明になった場合は、スキャンの再実行にはユニーク・インデックスが必要です。ユニーク・インデックスを作成しておかないと、スキャンがアボートする場合があります。

デフォルトでは、読み込み専用データベース上にないテーブルを独立性レベル 0 でスキャンする場合、ユニーク・インデックスが必要です。この要件を変更したい場合は、ユニークでないインデックスまたはテーブル・スキャンを Adaptive Server が選択するように、次のように指定します。

```
select * from table_name (index table_name)
```

基本となるテーブルを対象とした作業が行われると、スキャンが完了する前にアボートする場合があります。

繰り返し読み出し

繰り返し読み出しを実行するトランザクションは、トランザクション中に読み込むすべてのローまたはページをロックします。トランザクション内の 1 つのクエリがローを読み込んだ後、繰り返し読み出しトランザクションが完了されるまで、他のトランザクションはローを更新または削除できません。しかし、繰り返し読み出しトランザクションは、逐次トランザクションとは違って、範囲ロックを実行することによって幻を保護しません。他のトランザクションは値を挿入でき、繰り返し読み出しトランザクションによって読み込んだり、繰り返し読み出しトランザクションの検索基準で一致したローを更新したりできます。

繰り返し読み出しを実行するトランザクションは、トランザクション中に読み込むすべてのローまたはページをロックします。トランザクション内の 1 つのクエリがローを読み込んだ後、繰り返し読み出しトランザクションが完了されるまで、他のトランザクションはローを更新または削除できません。しかし、繰り返し読み出しトランザクションは、逐次トランザクションとは違って、範囲ロックを実行することによって幻を保護しません。他のトランザクションは値を挿入でき、繰り返し読み出しトランザクションによって読み込んだり、繰り返し読み出しトランザクションの検索基準で一致したローを更新したりできます。

注意　トランザクション独立性レベル 2 は、データオンリーロック・テーブルでのみサポートされます。全ページロック・テーブルでトランザクション独立性レベル 2 (繰り返し読み出し) を使用している場合は、独立性レベル 3 (逐次読み込み) が強制的に使用されます。

セッション・レベルで繰り返し読み出しを強制するには、次の文を使用します。

```
set transaction isolation level 2
```

または

```
set transaction isolation level repeatable read
```

クエリからトランザクション独立性レベル 2 を強制するには、次の文を使用します。

```
select title_id, price, advance
from titles
at isolation 2
```

または

```
select title_id, price, advance
from titles
at isolation repeatable read
```

トランザクション独立性レベル 2 は、トランザクション・レベルでのみサポートされます。独立性レベル 2 のクエリを設定するために、select または readtext 文で at isolation 句を使用することはできません。詳細については、「クエリの独立性レベルの変更」(676 ページ) を参照してください。

現在の独立性レベルのステータスの確認

グローバル変数 *@@isolation* は、Transact-SQL セッションの現在の独立性レベルを表します。*@@isolation* を問い合わせると、現在設定されているレベルの値 (0、1、または 3) が返されます。次に例を示します。

```
select @@isolation

--------
       1
```

クエリの独立性レベルの変更

select または readtext 文で at isolation 句を使用して、クエリの独立性レベルを変更します。at isolation 句は独立性レベル 0、1、3 をサポートします。独立性レベル 2 はサポートしていません。read uncommitted、read committed、serializable オプションは、次の独立性レベルをサポートします。

***at isolation* オプション**	**独立性レベル**
read uncommited	0
read committed	1
serializable	3

たとえば、次の 2 つの文では、同じテーブルに問い合わせます。上の文が独立性レベル 0、下の文がレベル 3 です。

```
select *
from titles
at isolation read uncommitted
select *
```

```
from titles
at isolation serializable
```

at isolation 句は、単一の select と readtext クエリに対してだけ、または declare cursor 文の中でだけ有効です。次のようなクエリで at isolation を使用した場合、Adaptive Server は構文エラーを返します。

- into 句を使用したクエリ
- サブクエリ内
- create view 文の中のクエリ
- insert 文の中のクエリ
- for browse 句を使用したクエリ

クエリ内に union 演算子がある場合は、最後の select の後に at isolation 句を指定する必要があります。

SQL-92 規格では、at isolation と set transaction isolation level のオプションとして、read uncommitted、read committed、serializable を定義しています。また、Transact-SQL 拡張機能によって、at isolation に 0、1、または 3 を指定できます。独立性レベルの説明を簡単にするために、このマニュアルの at isolation の例では、この拡張機能を使用していません。

また、select 文の holdlock キーワードを使用して、独立性レベル 3 を設定できます。ただし、at isolation read uncommitted も指定したクエリでは、noholdlock または shared を指定できません (クエリに holdlock キーワードと独立性レベル 0 を指定した場合、Adaptive Server は警告を発行し at isolation 句を無視します)。独立性レベルを設定するために複数の方法を使用する場合、holdlock キーワードは at isolation 句よりも優先されます (独立性レベル 0 の場合を除く)。また、at isolation は、set transaction isolation level が定義するセッション・レベルよりも優先されます。

『パフォーマンス&チューニング・シリーズ：ロックと同時実行制御』を参照してください。

独立性レベルの優先度

独立性レベルを定義する方法の違いによって、定義されたレベルの優先度がどのように変わるかを説明します。

1　holdlock、noholdlock、shared の各キーワードは、at isolation 句と set transaction isolation level オプションよりも優先されます。ただし、独立性レベルが 0 の場合は例外です。たとえば、次のように定義します。

```
/* This query executes at isolation level 3 */
select *
    from titles holdlock
    at isolation read committed
create view authors_nolock
    as select * from authors noholdlock
```

```
set transaction isolation level 3
/* This query executes at isolation level 1 */
select * from authors_nolock
```

2 at isolation 句は set transaction isolation level オプションよりも優先されます。次に例を示します。

```
set transaction isolation level 2
/* executes at isolation level 0 */
select * from publishers
    at isolation read uncommitted
```

at isolation 句の read uncommitted オプションは、holdlock、noholdlock、shared の各キーワードと同じクエリ内に指定できません。

3 set コマンドの transaction isolation level 0 オプションは、holdlock、noholdlock、shared の各キーワードよりも優先されます。次に例を示します。

```
set transaction isolation level 0
/* executes at isolation level 0 */
select *
    from titles holdlock
```

この例のクエリを実行しようとすると、Adaptive Server は警告を発行します。

カーソルと独立性レベル

Adaptive Server では、カーソルに対して 3 つの独立性レベルを設定できます。

- レベル 0 − 現在のカーソル位置を表すローを持つベース・テーブル・ページに対して、Adaptive Server はロックを適用しません。したがって、カーソルを使ったスキャン中でも読み込みロックが適用されないため、スキャンされているデータに対して他のアプリケーションからのアクセスがブロックされることはありません。ただし、独立性レベル 0 で機能しているカーソルは更新できません。また、このレベルでのスキャンの正確性を保証するには、ベース・テーブルにユニーク・インデックスが必要です。
- レベル 1 − 現在のカーソル位置を表すローを持つベース・テーブル・ページに対して、Adaptive Server は共有ロックまたは更新ロックを使用します。現在のカーソル位置が (fetch 文を実行した結果として) 別のページに移動するかカーソルがクローズしないかぎり、ページはロックされたままになります。インデックスを使ってベース・テーブルのローを検索している場合も、対応するインデックス・ページに共有ロックまたは更新ロックが使用されます。これは、Adaptive Server のデフォルトのロック動作です。

- レベル 3 – カーソルに代わってトランザクション内で読み込まれたベース・テーブル・ページすべてに対し、Adaptive Server は共有ロックまたは更新ロックを使用します。データ・ページが不要になるとロックは解除されるのではなく、トランザクションが終了するまで保持されます。holdlock キーワードを指定すると、テーブルまたはビューのクエリでの指定に従って、このロック・レベルがベース・テーブルに適用されます。

独立性レベル 2 は、カーソルではサポートされていません。

独立性レベル 3 で holdlock を使用するのではなく、set transaction isolation level を使用して、セッションに対する 4 つの独立性レベルのいずれかを指定できます。set transaction isolation level を使用する場合、トランザクション独立性レベルが 2 に設定されていなければ、オープンするカーソルはすべて指定した独立性レベルを使用します。この場合、カーソルは独立性レベル 3 を使用します。また、select 文の at isolation 句を使用して、特定のカーソルに独立性レベル 0、1、または 3 を指定できます。次に例を示します。

```
declare commit_crsr cursor
for select *
from titles
at isolation read committed
```

この文では、トランザクションやセッションの独立性レベルとは関係なく、カーソルが独立性レベル 1 で動作するように設定します。カーソルを独立性レベル 0 (read uncommitted) で宣言した場合、Adaptive Server はそのカーソルを読み込み専用として定義します。declare cursor 文では、at isolation read uncommitted と for update 句を一緒に指定できません。

カーソルを宣言するときではなく、オープンするときに、Adaptive Server はそのカーソルの独立性レベルを次の条件で決定します。

- カーソルが at isolation 句を使用して宣言されている場合、その独立性レベルは、カーソルをオープンするときのトランザクション独立性レベルを上書きする。
- カーソルが at isolation を使用して宣言されていない場合、カーソルは、オープンされたときの独立性レベルを使用する。カーソルをクローズして、再オープンすると、カーソルはトランザクションのその時点での独立性レベルになる。

カーソルを宣言すると、Adaptive Server はカーソルのクエリをコンパイルします。このコンパイル・プロセスは、独立性レベル 0 の場合と独立性レベル 1 または 3 の場合とでは異なります。独立性レベルを 1 か 3 に設定したトランザクション内に「言語」または「クライアント」カーソルを宣言した場合、そのカーソルを独立性レベル 0 のトランザクション内でオープンするとエラーになります。

次に例を示します。

```
set transaction isolation level 1
declare publishers_crsr cursor
    for select *
```

```
    from publishers
open publishers_crsr     /* no error */
fetch publishers_crsr
close publishers_crsr
set transaction isolation level 0
open publishers_crsr     /* error */
```

ストアド・プロシージャと独立性レベル

Sybase システム・プロシージャは、トランザクションやセッションの独立性レベルとは関係なく、常に独立性レベル 1 で動作します。ユーザ・ストアド・プロシージャは、そのプロシージャを実行するトランザクションの独立性レベルで動作します。独立性レベルがストアド・プロシージャ内で変更された場合、新しい独立性レベルはストアド・プロシージャを実行している間だけ有効です。

トリガと独立性レベル

データ修正文 (insert など) がトリガを起動するので、トランザクションの独立性レベルと独立性レベル 1 のうちどちらか高い方で、すべてのトリガを実行します。したがって、レベル 0 のトランザクションでトリガを起動した場合、Adaptive Server はトリガの独立性レベルを 1 に設定してから、最初の文を実行します。

SQL 規格への準拠

トランザクションを SQL 規格に準拠させるには、後続のトランザクションのモードと独立性レベルを変更する各アプリケーションの始めで、chained と transaction isolation level 3 オプションを**設定**する必要があります。カーソルを使用するアプリケーションの場合は、close on endtran オプションも設定してください。これらのオプションについては、「トランザクションでのカーソルの使用」(688 ページ) を参照してください。

パフォーマンスを向上させるための *lock table* コマンドの使用

lock table コマンドを使用すると、テーブルがアクセスされる前に明示的にロックを要求できるため、トランザクション中にそのテーブルにテーブル・ロックを設定できます。これは、即時テーブル・ロックが多くのローまたはページ・ロックの取得にかかるオーバヘッドを減らして、ロック時間が節約できるので便利です。次に、その例を示します。

- テーブルが同じトランザクションで 2 回以上スキャンされ、各スキャンが多くのページ・ロックまたはロー・ロックを必要とする場合。
- スキャンがテーブルのロックプロモーション・スレッショルドを超えるため、テーブル・ロックになるのがわかっている場合。

テーブル・ロックが明示的に要求されない場合、スキャンはテーブル・ロックを取得しようとするポイントでテーブルのロックプロモーション・スレッショルド (『リファレンス・マニュアル：プロシージャ』を参照) に達するまで、ページまたはロー・ロックを取得します。

lock table の構文は次のとおりです。

```
lock table table_name in {share | exclusive} mode
    [wait [no_of_seconds] | nowait]
```

wait/nowait オプションでは、ブロックされたテーブル・ロックを取得するのにコマンドが待機する時間を指定できます (詳細については、「lock table コマンドの wait/nowait オプション」(693 ページ) を参照してください)。

lock table コマンドを使用する場合の考慮事項は次のとおりです。

- lock table はトランザクション内でのみ発行できる。
- システム・テーブルでは lock table は使用できない。
- まず、lock table を使用して share モードでテーブルをロックし、次にそれを使用してロックを exclusive モードにアップグレードできる。
- 別々の lock table コマンドを使用して、同じトランザクション内で複数のテーブルをロックできる。
- テーブル・ロックが取得されると、lock table コマンドによってロックされたテーブルと、lock table コマンドのないロック・プロモーションによってロックされたテーブルに違いはない。

ストアド・プロシージャとトリガ内でのトランザクションの使用

文のバッチと同様に、ストアド・プロシージャとトリガ内でもトランザクションが使用できます。バッチまたはストアド・プロシージャ内のトランザクションが、トランザクションを含む別のストアド・プロシージャまたはトリガを呼び出す場合、呼び出されたトランザクションは最初のトランザクションにネストされます。

明示的または暗黙的な (連鎖モードの場合) 最初の begin transaction によって、バッチ、ストアド・プロシージャ、またはトリガ内でのトランザクションが開始されます。後続の begin transaction 文によって、ネスト・レベルの数が 1 つずつ増加します。後続の commit transaction 文によって、ネスト・レベルの数は、0 に達するまで 1 つずつ減少します。その後、トランザクション全体を Adaptive Server がコミットします。rollback transaction は、最初の begin transaction までトランザクション全体をアボートします。これはネスト・レベルや、それに含まれるストアド・プロシージャおよびトリガの数とは関係なく実行されます。

ストアド・プロシージャおよびトリガ内では、begin transaction 文の数と commit transaction 文の数は一致していなければなりません。連鎖モードを使用するストアド・プロシージャについても同じことが言えます。この場合、暗黙的にトランザクションを開始する最初の文にも、対応する commit transaction が必要になります。

ストアド・プロシージャ内の rollback transaction 文は、プロシージャまたはそのプロシージャの呼び出し元であるバッチの後続の文には影響しません。Adaptive Server は、ストアド・プロシージャまたはバッチの後続の文を実行します。ただし、トリガ内の rollback transaction 文はバッチ全体をアボートするので、バッチ内の後続の文は実行されません。

注意 トリガ内の rollback：1) トランザクションをロールバックし、2) トリガ内の後続の文を完了してから、3) バッチをアボートすることで、バッチ内の後続の文が実行されないようにします。

たとえば、次のバッチは、rollback transaction 文を含むストアド・プロシージャ myproc を呼び出します。

```
begin tran
update titles set ...
insert into titles ...
execute myproc
delete titles where ...
```

この場合、update と insert 文はロールバックされ、トランザクションはアボートされます。Adaptive Server は、バッチの実行を続け、delete 文を実行します。しかし、あるテーブルに rollback transaction 文を含む insert トリガが存在する場合は、バッチ全体がアボートされて delete 文は実行されません。次に例を示します。

```
begin tran
update authors set ...
insert into authors ...
delete authors where ...
```

ストアド・プロシージャに対して異なるトランザクション・モードまたは独立性レベルを設定するには、一定の要件を満たす必要があります。詳細については、「トランザクション・モードとストアド・プロシージャ」(685 ページ) を参照してください。トリガは常にデータ修正文の一部として呼び出されるため、現在のトランザクション・モードによる影響は受けません。

エラーとトランザクションのロールバック

データ整合性エラーが発生すると、暗黙的または明示的なトランザクションの状態が変化する場合があります。

- 重大度レベル 19 以上のエラー

  このようなエラーは、ユーザのサーバへの接続を終了させる結果となります。このため、ユーザ・トランザクション実行中にレベル 19 以上のエラーが発生した場合は、トランザクションがアボートされ、すべての文が begin transaction の最も外側までロールバックされます。Adaptive Server は、セッションの終わりで、コミットされていないトランザクションをロールバックします。

- データの整合性に影響を及ぼすエラーがデータ修正コマンドに存在する場合 (表 23-3 (684 ページ) 参照)

  - 算術オーバフロー・エラーおよびゼロ除算エラー (set arithabort arith_overflow コマンドを使用すると、トランザクションへの影響を変更できます)
  - パーミッション違反
  - ルール違反
  - 重複キー違反

rollback が、さまざまなコンテキスト内での Adaptive Server の処理に対してどのような影響を及ぼすかについて表 23-2 に示します。

表 23-2: 処理に対する rollback の影響

コンテキスト	*rollback* の影響
トランザクションのみ	トランザクションの開始後に修正されたすべてのデータがロールバックされる。トランザクションに複数のバッチがある場合、それらのバッチすべてが rollback の影響を受ける。 ロールバック後に発行されたコマンドはいずれも実行される。
ストアド・プロシージャのみ	なし
トランザクション内のストアド・プロシージャ	トランザクションの開始後に修正されたすべてのデータがロールバックされる。トランザクションに複数のバッチがある場合、それらのバッチすべてが rollback の影響を受ける。 ロールバック後に発行されたコマンドはいずれも実行される。 ストアド・プロシージャは、エラー・メッセージ 266「EXECUTE の後のトランザクション・カウントは、COMMIT か ROLLBACK TRAN がないことを示します。」を生成する。
トリガのみ	トリガは完了するが、トリガの結果はロールバックされる。 バッチ内の残りのコマンドは実行されない。次のバッチで処理が再開される。
トランザクション内のトリガ	トリガは完了するが、トリガの結果はロールバックされる。 トランザクションの開始後に修正されたすべてのデータがロールバックされる。トランザクションに複数のバッチがある場合、それらのバッチすべてが rollback の影響を受ける。 バッチ内の残りのコマンドは実行されない。次のバッチで処理が再開される。

コンテキスト	*rollback* の影響
ネストされたトリガ	内側のトリガは完了するが、トリガの結果はすべてロールバックされる。 バッチ内の残りのコマンドは実行されない。次のバッチで処理が再開される。
トランザクション内のネストされたトリガ	内側のトリガは完了するが、トリガの結果はすべてロールバックされる。 トランザクションの開始後に修正されたすべてのデータがロールバックされる。トランザクションに複数のバッチがある場合、それらのバッチすべてが rollback の影響を受ける。 バッチ内の残りのコマンドは実行されない。次のバッチで処理が再開される。

ストアド・プロシージャおよびトリガ内では、begin transaction 文の数と commit 文の数は一致していなければなりません。begin/commit 文がペアになっていないプロシージャまたはトリガを実行すると、警告メッセージが発行されます。連鎖モードを使用するストアド・プロシージャについても同じことが言えます。この場合、暗黙的にトランザクションを開始する最初の文にも、対応する commit 文が必要になります。

重複キー・エラーおよびルール違反があると、トリガは (return 文がないかぎり) 完了し、print、raiserror、リモート・プロシージャ・コール文などが実行されます。次に、トリガおよび残りのトランザクションはロールバックされ、残りのバッチはアボートされます。通常の SQL トランザクション (DB-Library の 2 フェーズ・コミットを使用しないもの) の内部から実行したリモート・プロシージャ・コールは、rollback 文によってロールバックされません。

表 23-3 に、重複キー・エラーまたはルール違反により生じるロールバックが、さまざまなコンテキスト内での Adaptive Server の処理に対してどのような影響を及ぼすかを示します。

表 23-3: 重複キー・エラー /ルール違反によるロールバック

コンテキスト	トランザクション中の変更エラーの影響
トランザクションのみ	現行のコマンドはアボートされる。前のコマンドはロールバックされず、後続のコマンドが実行される。
ストアド・プロシージャ内のトランザクション	同上
トランザクション内のストアド・プロシージャ	同上
トリガのみ	トリガは完了するが、トリガの結果はロールバックされる。 バッチ内の残りのコマンドは実行されない。次のバッチで処理が再開される。
トランザクション内のトリガ	トリガは完了するが、トリガの結果はロールバックされる。 トランザクションの開始後に修正されたすべてのデータがロールバックされる。トランザクションに複数のバッチがある場合、それらのバッチすべてがロールバックの影響を受ける。 バッチ内の残りのコマンドは実行されない。次のバッチで処理が再開される。
ネストされたトリガ	内側のトリガは完了するが、トリガの結果はすべてロールバックされる。 バッチ内の残りのコマンドは実行されない。次のバッチで処理が再開される。

コンテキスト	トランザクション中の変更エラーの影響
トランザクション内のネストされたトリガ	内側のトリガは完了するが、トリガの結果はすべてロールバックされる。 トランザクションの開始後に修正されたすべてのデータがロールバックされる。トランザクションに複数のバッチがある場合、それらのバッチすべてがロールバックの影響を受ける。 バッチ内の残りのコマンドは実行されない。次のバッチで処理が再開される。
ロールバック指定のトリガの後、トランザクション内にエラーがある場合	トリガの結果がロールバックされる。トランザクションの開始後に修正されたすべてのデータがロールバックされる。トランザクションに複数のバッチがある場合、それらのバッチすべてがロールバックの影響を受ける。 トリガは続行され、重複キーまたはルール・エラーが検出される。通常、トリガにより結果がロールバックされてトリガは続行されるが、この場合はトリガの結果はロールバックされない。 トリガが完了した後、バッチ内の残りのコマンドは実行されない。次のバッチで処理が再開される。

トランザクション・モードとストアド・プロシージャ

非連鎖トランザクション・モードを使用するストアド・プロシージャは、連鎖モードを使用するトランザクションとは互換性がありません。この逆の場合にも互換性はありません。次の例は、連鎖トランザクション・モードを使用する実行可能なストアド・プロシージャです。

```
create proc myproc
as
insert into publishers
    values ("9996", null, null, null)
commit work
```

非連鎖トランザクション・モードを使用するプログラムでこのプロシージャを呼び出すと、失敗します。これは、commit に対応する begin が存在しないためです。別の問題が発生する可能性もあります。

- 連鎖モードでトランザクションを開始するアプリケーションでは、長時間実行されるトランザクションが作成されたり、セッション全体のデータ・ロックを保持してしまうことがある。これらのことによって、Adaptive Server のパフォーマンスが低下する。
- アプリケーションによって、トランザクションのネストが予期しない場合に行われる可能性がある。この場合、使用しているトランザクション・モードによって結果が異なる。

一般的には、あるトランザクション・モードを使用するアプリケーションからは、それと同じモードを使用するストアド・プロシージャを呼び出す必要があります。このルールの例外として、Sybase システム・プロシージャ (sp_procxmode を除く) は、どちらのトランザクション・モードを使用したセッションからも呼び出すことができます。システム・プロシージャを実行するときにトランザクションがアクティブになっていないと、Adaptive Server はそのプロシージャが終了するまで、連鎖モードの設定を解除します。復帰する前に、モードが元の設定に戻されます。

Adaptive Server は、すべてのプロシージャを作成時のセッションのトランザクション・モード ("chained" または "unchained") でタグ付けします。これによって、あるモードを使用したトランザクションから別のモードを使用したトランザクションを呼び出す場合に発生する問題を回避できます。"chained" とタグ付けされたストアド・プロシージャは、非連鎖トランザクション・モードを使用するセッションでは実行できません。逆の場合も同じことが言えます。

トリガはトランザクション・モードでも実行できます。トリガは、常にデータ修正文の一部として呼び出されます。このため、連鎖トランザクション・モードを使用するセッションではトリガは連鎖トランザクションの一部となり、連鎖トランザクション・モードを使用しないセッションであればトリガの現在のトランザクション・モードが保持されます。

警告！ トランザクション・モードを使用する場合は、使用しているアプリケーションへの、それぞれの設定による影響に注意してください。

連鎖モードでのシステム・プロシージャの実行

Adaptive Server では、連鎖トランザクション・モードを使用するセッションで一部のシステム・プロシージャを実行できます。

- 次のシステム・プロシージャは、オープン・トランザクションが存在しない場合に連鎖トランザクション・モードを使用するセッションで実行できます。
  - sp_configure
  - sp_engine
  - sp_rename
- 次のシステム・プロシージャは、sp_procxmode を使用してトランザクション・モードを anymode に変更した後に、連鎖トランザクションを使用するセッションで実行できます。
  - sp_addengine
  - sp_dropengine
  - sp_showplan

- sp_sjobcontrol
- sp_sjobcmd
- sp_sjobcreate

『リファレンス・マニュアル：プロシージャ』を参照してください。

- sp_sjobdrop は連鎖トランザクション・モードを使用するセッションで実行できますが、オープン・トランザクション中に実行した場合は失敗します。

これらのストアド・プロシージャを実行すると、オープン・トランザクションが存在しない場合にこれらのストアド・プロシージャにより実行された変更が明示的にコミットされるため、commit または rollback を発行する必要があります。

以下を発行したときにオープン・トランザクションが存在すると、次のようになります。

- sp_rename、sp_configure、sp_engine、sp_addengine、または sp_dropengine − これらのプロシージャはトランザクション内で実行できないため、エラー 17260 で失敗します。
- sp_sjobcontrol、sp_sjobcmd、sp_sjobcreate、sp_sjobdrop、または sp_showplan − プロシージャの実行後トランザクションが開いたままになります。トランザクション全体に対して commit または rollback を明示的に発行する必要があります。

  これらのプロシージャの実行時にエラーが出されると、プロシージャ内で実行された操作のみがロールバックします。同じトランザクション内で操作が実行されている場合でも、プロシージャ実行前に行われた操作がロールバックすることはありません。

set chained {on | off} を使用してセッションの連鎖モードを設定します。『リファレンス・マニュアル：コマンド』を参照してください。

ストアド・プロシージャのトランザクション・モードの設定

sp_procxmode を使用すると、ストアド・プロシージャのトランザクション・モードを表示または変更できます。たとえば、ストアド・プロシージャ byroyalty のトランザクション・モードを連鎖に変更するには、次のように入力します。

```
sp_procxmode byroyalty, "chained"
```

sp_procxmode “anymode” に設定すると、ストアド・プロシージャを連鎖および非連鎖のどちらのトランザクション・モードでも実行できます。次に例を示します。

```
sp_procxmode byroyalty, "anymode"
```

sp_procxmode にパラメータ値を指定しないで実行すると、現在のデータベース内のすべてのストアド・プロシージャのトランザクション・モードが表示されます。

```
sp_procxmode

procedure name               transaction mode
-------------------------    --------------------
byroyalty                    Any Mode
discount_proc                Unchained
history_proc                 Unchained
insert_sales_proc            Unchained
insert_salesdetail_proc      Unchained
storeid_proc                 Unchained
storename_proc               Unchained
title_proc                   Unchained
titleid_proc                  Unchained
```

sp_procxmode は、非連鎖トランザクション・モードでだけ使用できます。

プロシージャのトランザクション・モードを変更できるのは、システム管理者、データベース所有者、プロシージャ所有者だけです。

トランザクションでのカーソルの使用

デフォルトでは、commit または rollback によるトランザクション終了の時点で、Adaptive Server はカーソルのステータス (オープンまたはクローズ) を変更しません。しかし、SQL 規格では、オープンしているカーソルと現在実行中のトランザクションは連動しています。トランザクションをコミットまたはロールバックすると、それと関連するオープンしているカーソルも自動的にクローズされます。

この SQL 規格に準拠するため、Adaptive Server では set コマンドの close on endtran オプションを使用できます。さらに、連鎖モードを on に設定している場合、カーソルをオープンすると Adaptive Server によってトランザクションが開始され、最も外側のトランザクションがコミットまたはロールバックされるとそのカーソルがクローズされます。

たとえば、デフォルトで次の文を実行するとエラーが発生します。

```
open test_crsr
commit tran
open test_crsr
```

close on endtran または chained オプションを on にしている場合、最も外側のトランザクションがコミットされた後カーソルのステータスがオープンからクローズに変更されます。この場合、カーソルを再びオープンできます。

注意 クライアント・アプリケーションのバッファのローはカーソルを介して戻されるので、ユーザはバッファ内でスクロールもできます。ただし、トランザクションがアボートしたときはクライアント・アプリケーションでの逆方向のスクロールは避けるようにしてください。close on endtran オプションまたは連鎖モードによるトランザクションのロールバック (クライアント側では認識されない) が行われるため、クライアント・キャッシュ内のローが無効になっている場合があります。

トランザクション内のカーソルが取得した排他ロックはすべて、そのトランザクションが終了するまで保持されます。これは、holdlock キーワード、at isolation serializable 句、または set isolation level 3 オプションを使用する場合は、共有ロックにも当てはまります。

トランザクションに関してカーソルを使って行われる更新の動作は、次の規則によって決まります。

- 明示的なトランザクション内で行われた更新は、トランザクションの一部とみなされます。トランザクションがコミットされると、トランザクション内で行われた更新もすべてコミットされます。トランザクションがアボートされると、トランザクション内で行われた更新がすべてロールバックされます。アボートされたトランザクションの外部で同一のカーソルを使って行われた更新には影響しません。
- 明示的な (しかもクライアントで指定された) トランザクション内で、カーソルを使って更新が行われた場合、カーソルがクローズされたときに Adaptive Server はそれらの更新をコミットしません。カーソルに対応するトランザクションが終了した場合にかぎり、保留中の更新がコミットまたはロールバックされます。
- トランザクションがコミットまたはアボートされても、結果のローを操作しない SQL カーソル文 (declare cursor、open cursor、close cursor、set cursor rows、deallocate cursor) には影響しません。たとえば、クライアントがトランザクション内でカーソルをオープンし、そのトランザクションがアボートした場合、カーソルはそのあともオープンしたままになります (ただし、close on endtran オプションが設定されているか、連鎖トランザクション・モードが使用されている場合は除く)。

ただし、close on endtran オプションを設定しないと、トランザクションが終了した後もカーソルはオープンしたままで、その現在のページ・ロックは有効なままになります。カーソルが他のローをフェッチすると、ロックを取得し続ける場合もあります。

トランザクションを使用する場合の考慮事項

アプリケーション内にトランザクションを使用する場合は、次に示す事項を考慮してください。

- トランザクション名またはセーブポイント名を指定しないで rollback 文を使用すると、すべての文は最も外側の (明示的または暗黙的な) begin transaction 文までロールバックされ、トランザクションはキャンセルされます。rollback の発行時に、現行のトランザクションがない場合、文には効力がありません。

  トリガまたはストアド・プロシージャ内で、トランザクション名またはセーブポイント名を指定しないで rollback 文を使用すると、すべての文は最も外側の (明示的または暗黙的な) begin transaction 文までロールバックされます。

- rollback 文を使用しても、ユーザに対するメッセージは生成されません。警告を表示する必要がある場合は、raiserror または print 文を使用します。

- 多数の Transact-SQL コマンドをグループ化し、実行時間の長い 1 つのトランザクションにすると、リカバリ時間に影響することがあります。長時間のトランザクション実行中に Adaptive Server に障害が発生すると、Adaptive Server はまずトランザクション全体を取り消すので、リカバリ時間が増加します。

- ユーザ・トランザクション内のデータベースは、Adaptive Server にインストールされているデータベースと同じ数まで使用できます。たとえば、Adaptive Server に 25 のデータベースがある場合は、ユーザ・トランザクションに 25 のデータベースを含めることができます。

- リモート・プロシージャ・コール (RPC) は、呼び出し元のトランザクションとは別に実行することができます。標準トランザクション (Open Client の DB-Library/C 2 フェーズ・コミットまたは Adaptive Server 分散トランザクション管理機能を使用しないトランザクション) では、RPC を介してリモート・サーバにより実行されたコマンドは、rollback によってロールバックされず、実行対象の commit に依存しません。

- クライアント・アプリケーションとサーバの間の接続が複数にわたるトランザクションは実行できません。たとえば DB-Library/C アプリケーションで、SQL 文を複数のープンされている DBPROCESS 接続にわたる 1 つのトランザクションにグループ化することはできません。

- Adaptive Server では、ログに対して 2 つのスキャンが実行されます。最初のスキャンではデータ・ページの割り付け解除と予約されていないページを探し、2 番目のスキャンではログ・ページの割り付け解除を探します。これらのスキャンは内部の最適化なのでユーザからは見えません。スキャンは自動的に実行され、スキャンのオン・オフを切り替えることはできません。

コミット後の最適化では、Adaptive Server はこれらのログ・レコードが含まれている後方の「次の」ログ・ページを記憶します。コミット後の段階で、Adaptive Server は、ページのレコードの処理後に、コミット後の作業が必要な「次の」ページに移動します。多数のユーザが同時にトランザクションのログを syslogs に記録する並列処理環境では、コミット後の最適化により不要なログ・ページの読み込みやスキャンが回避され、コミット後の処理のパフォーマンスが向上します。

最適化は、診断情報に表示されません。

トランザクションのバックアップとリカバリ

データベースが変更されると、それが単一の update 文によるものであるか、複数の SQL 文のセットによるものであるかに関係なく、それぞれの変更が syslogs システム・テーブルに記録されます。このテーブルは「トランザクション・ログ」と呼ばれます。

truncate table、インデックスのないテーブルへの一括コピー、select into, writetext、dump transaction with no_log など、データベースを変更するコマンドによっては、ログに記録されないものもあります。

update、insert、delete 文は、実行されるたびにトランザクション・ログに記録されます。トランザクションが開始されると、begin transaction イベントがログに記録されます。データ修正文は、実行されるたびにログへの記録が行われます。

データベース自体の変更が実行される前に、変更は、常にログに記録されます。このタイプのログは先書きログと呼ばれ、データベースに障害が発生した場合の完全なリカバリを可能にします。

障害の原因は、ハードウェアまたはメディア問題、システム・ソフトウェアの問題、アプリケーション・ソフトウェアの問題、プログラムが原因のトランザクションの中断、ユーザによる中断などが考えられます。

どの障害も、dump コマンドで作成されたバックアップからリストアされたデータベースのコピーに対して、トランザクション・ログに記録されている内容を再現できます。

障害からリカバリするには、障害発生時に実行中で、まだコミットしていないトランザクションを取り消す必要があります。部分的なトランザクションは正しい変更ではないためです。完了したトランザクションは、データベース・デバイスに完全に記録されているという保証がないかぎり、再度実行する必要があります。

実行時間の長いトランザクションが実行中で、まだコミットされていないときに Adaptive Server に障害が発生すると、変更を取り消すには、トランザクションの実行にかかった時間と同じくらいの時間がかかる場合があります。このような例として、begin transaction に対応する commit transaction または rollback transaction が指定されていないトランザクションがあります。このような場合、Adaptive Server は変更の書き込みができなくなり、リカバリに時間がかかります。

Adaptive Server の動的ダンプによって、データベースが使用されている間もデータベースおよびトランザクション・ログのバックアップを行うことができます。データベース・トランザクション・ログは頻繁にバックアップをとってください。データを頻繁にバックアップするほど、システムに障害が発生した場合に失う作業の量は少なくなります。

各データベースの所有者または ss_oper という役割を持つユーザは、oper_role コマンドを使用してデータベースおよびそのトランザクション・ログをバックアップする責任があります。このコマンドの実行パーミッションは、デフォルトでデータベース所有者に付与されますが、他のユーザに譲渡できます。しかし、load コマンドを使用するパーミッションは、デフォルトではデータベース所有者に付与されますが、譲渡はできません。

load コマンドを適切に実行すると、Adaptive Server はすべてのリカバリ処理を実行します。Adaptive Server は、チェックポイント・インターバルも制御します。チェックポイントは、変更のあったデータ・ページすべてがデータベース・デバイスに書き込まれるように保証されているポイントです。ユーザは必要に応じて、checkpoint コマンドでチェックポイントを設定できます。

バックアップとリカバリの詳細については、『リファレンス・マニュアル：コマンド』と『システム管理ガイド 第 2 巻』の「第 11 章 バックアップおよびリカバリ・プランの作成」を参照してください。

第 24 章　ロックのコマンドとオプション

トピック名	ページ
ロックを待機する時間制限の設定	693
キュー処理のための読み飛ばしロック	696

ロックを待機する時間制限の設定

Adaptive Server では、コマンドがロックを取得するためにどの程度の時間待機するのかを決定するロック待機時間を次のように指定できます。

- lock table コマンドの wait オプションまたは nowait オプションを使用して、テーブル・ロックを取得するために待機する時間制限を指定できます。
- セッションにおいては、セッション中に発行されるすべての後続コマンドに対してロック待機時間を指定するために、set lock コマンドが使用できます。
- セッションレベルの設定 set lock wait nnn とともに使用する sp_configure のパラメータ lock wait period は、ユーザ定義テーブルにのみ適用できます。これらの設定は、システム・テーブルには影響しません。
- ストアド・プロシージャにおいては、ストアド・プロシージャ内で発行されるすべての後続コマンドに対してロック待機時間を指定するために、set lock コマンドが使用できます。
- サーバワイドのロック待機時間を設定するには、sp_configure の lock wait period オプションが使用できます。

lock table コマンドの *wait*/*nowait* オプション

トランザクションの中で、lock table コマンドを使うと、テーブル・ロックに拡大するために十分なローレベルやページレベルのロックをコマンドが取得するまで待たなくても、テーブルに対するテーブル・ロックを要求できます。

lock table コマンドには wait/nowait オプションがあり、これによって、他のトランザクションのオペレーションがターゲット・テーブルに保持しているロックを放棄するまでこのコマンドが待つ時間の長さを指定できます。

lock table の構文は次のとおりです。

```
lock table table_name in {share | exclusive} mode
    [wait [no_of_seconds] | nowait]
```

トランザクションの内部にある次のコマンドは、titles テーブルに対するテーブル・ロックを取得するための 2 秒間の待機時間を設定します。

```
lock table titles in share mode wait 2
```

テーブル・ロックを取得する前に待機時間が経過した場合、トランザクションは続行され、lock table が使用されなかったときと同じようにロー・ロックまたはページ・ロックが使用され、次に示す情報メッセージ (エラー番号 12207) が生成されます。

```
Could not acquire a lock within the specified wait
period.COMMAND level wait...
```

トランザクション中にこのエラー・メッセージを処理するコード例については、『リファレンス・マニュアル：コマンド』を参照してください。

注意 *no_of_seconds* を指定しないで lock table...wait を使用する場合、コマンドはロックを無制限に待機します。

これ以降の項で説明するように、ロックの待機時間の時間制限は、セッション・レベルとシステム・レベルで設定できます。lock table コマンドを使用して設定する待機時間は、これら両方の時間制限を無効にします。

nowait オプションは、0 秒の待機時間が設定された wait オプションと同じです。lock table は即時にテーブル・ロックを取得するか、または上記で説明した情報メッセージを生成します。ロックを取得できない場合、トランザクションは lock table コマンドが使用されなかったときと同じように続行されます。

セッション・レベルまたはストアド・プロシージャ内のいずれかで set lock コマンドを使用して、ロックを取得するためのタスクの待機時間を制御できます。

システム管理者は、sp_configure オプションの lock wait period を使って、ロックの取得に関するサーバワイドの時間制限を設定できます。

セッションレベルのロック待機時間の設定

set lock wait を使用すると、セッションまたはストアド・プロシージャ内でコマンドがロックを取得するための待機時間を制御できます。構文は次のとおりです。

```
set lock {wait no_of_seconds | nowait}
```

no_of_seconds は整数で入力します。たとえば、次の例は、ロックの待機時間として 5 秒間のセッションレベル時間制限を設定します。

```
set lock wait 5
```

例外が 1 つありますが、コマンドがロックを取得する前に set lock wait の期間が経過した場合、コマンドは失敗し、そのコマンドを含むトランザクションはロールバックされ、次のエラー・メッセージが生成されます。

```
Msg 12205, Level 17, State 2:
Server 'sagan', Line 1:
Could not acquire a lock within the specified wait
period.SESSION level wait period=300 seconds, spid=12, lock
type=shared page, dbid=9, objid=2080010441, pageno=92300,
rowno=0.Aborting the transaction.
```

例外は、トランザクション内の lock table が set lock wait よりも長い待機時間を設定した場合です。その場合、前述の項で示したように、トランザクションはタイムアウトするまで lock table の待機時間を使用します。

set lock nowait オプションは、0 秒の待機時間が設定された set lock wait オプションと同じです。lock table 以外のコマンドが要求したロックを即座に取得できない場合、そのコマンドは失敗し、トランザクションはロールバックされ、前述のエラー・メッセージが生成されます。

サーバワイドのロック待機時間とセッションレベルのロック待機時間の両方が設定されている場合、セッションレベルの待機時間の方が優先されます。セッションレベルの待機時間が何も設定されていない場合、サーバレベルの待機時間が使用されます。

サーバワイドのロック待機時間の設定

システム管理者は、設定パラメータ lock wait period でサーバワイドのロック待機時間を設定できます。

コマンドがロックを取得する前にロック待機時間が経過した場合、これを無効にする set lock wait または lock table 待機時間が存在しないかぎり、コマンドは失敗し、コマンドを含むトランザクションはロールバックされ、次のエラー・メッセージが生成されます。

```
Msg 12205, Level 17, State 2:
Server 'wiz', Line 1:
Could not acquire a lock within the specified wait
period.SERVER level wait period=300 seconds, spid=12, lock
type=shared page, dbid=9, objid=2080010441, pageno=92300,
```

```
rowno=0.Aborting the transaction.
```

set lock wait または lock table wait によって入力される時間制限は、サーバレベルのロック待機時間を無効にします。このため、たとえば、サーバレベルの待機時間が 5 秒で、かつセッションレベルの待機時間が 10 秒である場合、update コマンドはロックを取得するまで 10 秒間待ち、それでもロックを取得できなかった場合、コマンドは失敗し、トランザクションはアボートされます。

デフォルトのサーバレベルのロック待機時間は、事実上「永久に待機」します。時間制限のある待機を設定した後でデフォルト値に戻すには、次のように sp_configure を使用して lock wait period の値を設定します。

```
sp_configure "lock wait period", 0, "default"
```

ロック待機タイムアウトの数値についての情報

sp_sysmon は、ロックを待っていたタスクが、指定された時間内にロックを取得できなかった回数についてレポートします。

キュー処理のための読み飛ばしロック

読み飛ばしロックは、select コマンドと readtext コマンド、およびデータ修正コマンドの update、delete、writetext で使用できるオプションです。読み飛ばしロックは、コマンドが検出する非両立ロックすべてを、ブロックしたり、終了したり、メッセージを生成したりすることなく、暗黙的に省略するようにコマンドに指示します。テーブルのローがキューを構成するときには、これが主に使われます。キューイングされたロー (たとえばキューイングされた顧客や顧客の注文を表す場合もあります) を処理するために、多くのタスクがテーブルにアクセスすることがあります。所定のタスクは、キューの特定のメンバを処理することに関係しませんが、その選択基準を満たす、キューの中の使用できるメンバを処理することに関係します。

読み飛ばしクエリ中の非両立ロック

select コマンドと readtext コマンドでは、非両立ロックとは排他ロックのことです。このため、select コマンドと readtext コマンドは、共有ロックや更新ロックが保持されている任意のローやページにアクセスできます。

delete、update、writetext コマンドでは、すべての種類のページ・ロックおよびロー・ロックは両立しないので、次のようになります。

- 共有、更新、または排他のロー・ロックを伴うすべてのローは、データローロック・テーブルでは省略される。

- 共有、更新、または排他のロックを伴うすべてのページは、データページロック・テーブルでは省略される。

readpast を指定するすべてのコマンドは、排他的テーブル・ロックが存在する場合、トランザクション独立性レベル 0 で実行される select コマンドを除いて、ブロックされます。

全ページロック・テーブルと読み飛ばしクエリ

readpast オプションを全ページロック・テーブルに指定した場合、readpast オプションは無視されます。コマンドは、コマンドまたはセッションに対して指定された独立性レベルで実行されます。

- 独立性レベルが 0 の場合、ダーティ・リードが実行され、コマンドはロックされたローから値を返し、ブロックしない。
- 独立性レベルが 1 または 3 の場合、互換性のないロックを伴うページを読み込む必要があるときにコマンドがブロックします。

読み飛ばしを伴う *select* クエリにおける独立性レベルの影響

読み飛ばしロックは、トランザクション独立性レベル 1 または 2 で使用するように設計されています。

セッションレベルのトランザクション独立性レベルと読み飛ばし

データオンリーロック・テーブルにおいて、select コマンドでの readpast のテーブルに対する影響を表 24-1 に示します。

表 24-1: セッションレベルの独立性レベルと読み飛ばしの使用

セッション独立性レベル	影響
0、read uncommitted (ダーティ・リード)	readpast は無視され、コミットされていないトランザクションを含むローがユーザに返される。警告メッセージが出力される。
1、read committed	非両立ロックを伴うローやページは省略される。つまり、読み込まれたローやページにはロックが保持されない。
2、repeatable read	非両立ロックを伴うローやページは省略される。つまり、読み込まれたローやページに対する共有ロックは、文やトランザクションが完了するまで保持される。
3、serializable	readpast は無視され、コマンドはレベル 3 で実行される。コマンドは、非両立ロックを伴うすべてのローまたはページをブロックする。

クエリレベルの独立性レベルと読み飛ばし

readpast を指定する select コマンドが次の句のいずれかを含む場合、コマンドは失敗し、エラー・メッセージが表示されます。

- 0 または read uncommitted を指定する at isolation 句
- 3 または serializable を指定する at isolation 句
- 同じテーブルに対する holdlock キーワード

readpast を指定する select クエリが at isolation 2 または at isolation repeatable read も指定する場合、文やトランザクションが完了するまで readpast テーブルに対して共有ロックが保持されます。

readpast を含み、at isolation read uncommitted を指定する readtext コマンドは、警告メッセージを発行してから独立性レベル 0 で自動的に実行されます。

readpast 付きのデータ修正コマンドと独立性レベル

セッションのトランザクション独立性レベルが 0 の場合、readpast を使う delete、update、writetext コマンドは警告メッセージを発行しません。

- データページロック・テーブルに対して、これらのコマンドは非両立ロックでロックされていないすべてのページのすべてのローを修正する。
- データローロック・テーブルに対して、これらのコマンドは非両立ロックでロックされていないすべてのローに影響を与える。

セッションのトランザクション独立性レベルが 3 (逐次読み取り) の場合、readpast を使う delete、update、writetext コマンドは、非両立ロックを伴うローまたはページを検出すると自動的にブロックします。

トランザクション独立性レベル 2 (逐次読み取り) では、delete、update、writetext コマンドは次のことを実行します。

- 非両立ロックでロックされていないすべてのページのすべてのローを修正する。
- データローロック・テーブルに対して、これらのコマンドは非両立ロックでロックされていないすべてのローに影響を与える。

text、*unitext*、*image* カラムと読み飛ばし

readpast オプション付きの select コマンドが、非両立ロックが設定された text カラムを検出した場合、読み飛ばしロックはローを取り出しますが、値が null の text カラムを返します。この場合、カラムがロックされているので、null 値が格納されている text カラムと、戻された null 値は区別されません。

readpast オプション付きの update コマンドが 2 つ以上の text カラムに適用され、そのときにチェックされた最初の text カラムに非両立ロックが保持されている場合、読み飛ばしロックはそのローを省略します。カラムに互換性のないロックが設定されていない場合、コマンドはロックを取得し、カラムを修正します。ローの後続の text カラムに互換性のないロックが設定されている場合、ロックを取得し、カラムを修正できるまで、コマンドはブロックします。

readpast オプション付きの delete コマンドは、ロー内の text カラムに非両立ロックが保持されている場合、ローを省略します。

読み飛ばしロックの例

次に読み飛ばしロックの例を示します。

排他ロックが保持されているすべてのローを省略する場合は、次を実行します。

```
select * from titles readpast
```

他のセッションによるロックが保持されていないローだけを更新する場合は、次を実行します。

```
update titles
    set price = price * 1.1
    from titles readpast
```

titles テーブルには読み飛ばしロックを使い、authors テーブルや titleauthor テーブルには使わない場合は、次を実行します。

```
select *
    from titles readpast, authors, titleauthor
    where titles.title_id = titleauthor.title_id
    and authors.au_id = titleauthor.au_id
```

stores テーブルではロックされていないローだけを削除し、スキャンには authors テーブルをブロックすることを許可する場合は、次を実行します。

```
delete stores from stores readpast, authors
where stores.city = authors.city
```

付 録　A

pubs2 データベース

この付録では、サンプル・データベース pubs2 について説明します。このサンプル・データベースには、publishers、authors、titles、titleauthor、salesdetail、sales、stores、roysched、discounts、blurbs、および au_pix の各テーブルが含まれています。

pubs2 データベースには、これらのテーブルを作成するために使用するプライマリ・キーと外部キー、ルール、デフォルト、ビュー、トリガ、ストアド・プロシージャも含まれます。

pubs2 データベースの関係図は図 A-1 (710 ページ) にあります。

pubs2 のインストールについては、使用しているプラットフォームの『インストール・ガイド』を参照してください。

create やデータ修正文を使用してサンプル・データベースを変更するには、システム管理者から追加のパーミッションを取得する必要があります。サンプル・データベースを変更する場合は、次のユーザが使用するときのために、データベースを元の状態に戻すことをおすすめします。サンプル・データベースのリストアにヘルプが必要な場合は、システム管理者に連絡してください。

pubs2 データベース内のテーブル

pubs2 データベースの各テーブル内のカラム・ヘッダは、カラム名、そのデータ型 (ユーザ定義データ型も含む)、その null または not null のステータスを示しています。また、カラム・ヘッダは、カラムに影響を与えるデフォルト、ルール、トリガ、インデックスも示しています。

publishers テーブル

publishers テーブルの内容は、各出版社の名前、ID、所在都市と州です。

publishers は次のように定義されています。

```
create table publishers
 (pub_id char (4) not null,
pub_name varchar(40) not null,
city varchar(20) null,
state char(2) null)
```

プライマリ・キーは pub_id です。

```
sp_primarykey publishers, pub_id
```

pub_idrule ルールは次のように定義されています。

```
create rule pub_idrule
as @pub_id in
("1389", "0736", "0877", "1622", "1756")
or @pub_id like "99[0-9][0-9]"
```

authors テーブル

authors テーブルの内容は、作家の名前、電話番号、作家 ID、その他の情報です。

authors は次のように定義されています。

```
create table authors
(au_id id not null,
au_lname varchar(40) not null,
au_fname varchar(20) not null,
phone char(12) not null,
address varchar(40) null,
city varchar(20) null,
state char(2) null,
country varchar(12) null,
postalcode char(10) null)
```

プライマリ・キーは au_id です。

```
sp_primarykey authors, au_id
```

au_lname カラムと au_fname カラムのノンクラスタード・インデックスは次のように定義されています。

```
create nonclustered index aunmind
on authors (au_lname, au_fname)
```

phone カラムには以下のデフォルトが使用されています。

```
create default phonedflt as "UNKNOWN"
sp_bindefault phonedft, "authors.phone"
```

次のビューは authors を使用しています。

```
create view titleview
as
select title, au_ord, au_lname,
price, total_sales, pub_id
from authors, titles, titleauthor
where authors.au_id = titleauthor.au_id
and titles.title_id = titleauthor.title_id
```

titles テーブル

titles テーブルの内容は、タイトルの ID、タイトル、種類、出版社 ID、価格、その他の情報です。

titles は次のように定義されています。

```
create table titles
(title_id tid not null,
title varchar(80) not null,
type char(12) not null,
pub_id char(4) null,
price money null,
advance money null,
total_sales int null,
notes varchar(200) null,
pubdate datetime not null,
contract bit not null)
```

プライマリ・キーは title_id です。

```
sp_primarykey titles, title_id
```

pub_id カラムは、publishers テーブルに対する外部キーです。

```
sp_foreignkey titles, publishers, pub_id
```

title カラムのノンクラスタード・インデックスは次のように定義されています。

```
create nonclustered index titleind
on titles(title)
```

title_idrule は次のように定義されています。

```
create rule title_idrule
as
@title_id like "BU[0-9][0-9][0-9][0-9]" or
@title_id like "[MT]C[0-9][0-9][0-9][0-9]" or
@title_id like "P[SC][0-9][0-9][0-9][0-9]" or
@title_id like "[A-Z][A-Z]xxxx" or
@title_id like "[A-Z][A-Z]yyyy"
```

type カラムには以下のデフォルトが使用されています。

```
create default typedflt as "UNDECIDED"
sp_bindefault typedflt, "titles.type"
```

pubdate カラムには以下のデフォルトが設定されています。

```
create default datedflt as getdate()
sp_bindefault datedflt, "titles.pubdate"
```

titles は次のトリガを使用します。

```
create trigger deltitle
on titles
for delete
as
```

```
if (select count(*) from deleted, salesdetail
where salesdetail.title_id = deleted.title_id) >0
begin
   rollback transaction
   print "You can't delete a title with sales."
end
```

次のビューは titles を使用しています。

```
create view titleview
as
select title, au_ord, au_lname,
price, total_sales, pub_id
from authors, titles, titleauthor
where authors.au_id = titleauthor.au_id
and titles.title_id = titleauthor.title_id
```

titleauthor テーブル

titleauthor テーブルの内容は、作家 ID、タイトル ID、タイトルの印税 (パーセンテージ) です。

titleauthor は次のように定義されています。

```
create table titleauthor
(au_id id not null,
title_id tid not null,
au_ord tinyint null,
royaltyper int null)
```

プライマリ・キーは au_id と title_id です。

```
sp_primarykey titleauthor, au_id, title_id
```

title_id カラムおよび au_id カラムは titles と authors に対する外部キーです。

```
sp_foreignkey titleauthor, titles, title_id
sp_foreignkey titleauthor, authors, au_id
```

au_id カラムのノンクラスタード・インデックスは次のように定義されています。

```
create nonclustered index auidind
on titleauthor(au_id)
```

title_id カラムのノンクラスタード・インデックスは次のように定義されています。

```
create nonclustered index titleidind
on titleauthor(title_id)
```

次のビューは titleauthor を使用しています。

```
create view titleview
as
select title, au_ord, au_lname,
price, total_sales, pub_id
from authors, titles, titleauthor
where authors.au_id = titleauthor.au_id
and titles.title_id = titleauthor.title_id
```

次のプロシージャでは、titleauthor を使用します。

```
create procedure byroyalty @percentage int
as
select au_id from titleauthor
where titleauthor.royaltyper = @percentage
```

salesdetail テーブル

salesdetail テーブルの内容は、書店 ID、注文 ID、タイトル番号、販売冊数、および販売値引きです。

salesdetail は次のように定義されています。

```
create table salesdetail
(stor_id char(4) not null,
ord_num numeric(6,0)
title_id tid not null,
qty smallint not null,
discount float not null)
```

プライマリ・キーは stor_id と ord_num です。

```
sp_primarykey salesdetail, stor_id, ord_num
```

title_id カラム、stor_id カラム、ord_num カラムは titles と sales に対する外部キーです。

```
sp_foreignkey salesdetail, titles, title_id
sp_foreignkey salesdetail, sales, stor_id, ord_num
```

title_id カラムのノンクラスタード・インデックスは次のように定義されています。

```
create nonclustered index titleidind
on salesdetail (title_id)
```

stor_id カラムのノンクラスタード・インデックスは次のように定義されています。

```
create nonclustered index salesdetailind
on salesdetail (stor_id)
```

title_idrule ルールは次のように定義されています。

```
create rule title_idrule
as
@title_id like "BU[0-9][0-9][0-9][0-9]" or
@title_id like "[MT]C[0-9][0-9][0-9][0-9]" or
@title_id like "P[SC][0-9][0-9][0-9][0-9]" or
@title_id like "[A-Z][A-Z]xxxx" or
@title_id like "[A-Z][A-Z]yyyy"
```

salesdetail は次のトリガを使用します。

```
create trigger totalsales_trig on salesdetail
    for insert, update, delete
as
/* Save processing:  return if there are no rows affected */
if @@rowcount = 0
    begin
      return
end
/* add all the new values */
/* use isnull:  a null value in the titles table means
**              "no sales yet" not "sales unknown"
*/
update titles
    set total_sales = isnull(total_sales, 0) + (select
sum(qty)
    from inserted
    where titles.title_id = inserted.title_id)
    where title_id in (select title_id from inserted)
/* remove all values being deleted or updated */
update titles
    set total_sales = isnull(total_sales, 0) - (select
sum(qty)
    from deleted
    where titles.title_id = deleted.title_id)
    where title_id in (select title_id from deleted)
```

sales テーブル

sales テーブルの内容は、販売書店 ID、注文番号、販売日です。

sales は次のように定義されています。

```
create table sales
(stor_id char(4) not null,
ord_num varchar(20) not null,
date datetime not null)
```

プライマリ・キーは stor_id と ord_num です。

```
sp_primarykey sales, stor_id, ord_num
```

stor_id カラムは、stores に対する外部キーです。

```
sp_foreignkey sales, stores, stor_id
```

stores テーブル

stores テーブルの内容は、書店の名前、住所、ID 番号、支払期限です。

stores は次のように定義されています。

```
create table stores
(stor_id char(4) not null,
stor_name varchar(40) not null,
stor_address varchar(40) null,
city varchar(20) null,
state char(2) null,
country varchar(12) null,
postalcode char(10) null,
payterms varchar(12) null)
```

プライマリ・キーは stor_id です。

```
sp_primarykey stores, stor_id
```

roysched テーブル

roysched テーブルの内容は、価格のパーセンテージとして定義される印税です。

roysched は次のように定義されています。

```
create table roysched
title_id tid not null,
lorange int null,
hirange int null,
royalty int null)
```

プライマリ・キーは title_id です。

```
sp_primarykey roysched, title_id
```

title_id カラムは、titles に対する外部キーです。

```
sp_foreignkey roysched, titles, title_id
```

title_id カラムのノンクラスタード・インデックスは次のように定義されています。

```
create nonclustered index titleidind
on roysched (title_id)
```

discounts テーブル

discounts テーブルの内容は、書店での値引きです。

discounts は次のように定義されています。

```
create table discounts
(discounttype varchar(40) not null,
stor_id char(4) null,
lowqty smallint null,
highqty smallint null,
discount float not null)
```

プライマリ・キーは discounttype と stor_id です。

```
sp_primarykey discounts, discounttype, stor_id
```

stor_id は、stores に対する外部キーです。

```
sp_foreignkey discounts, stores, stor_id
```

blurbs テーブル

blurbs テーブルの内容は、本の広告サンプルです。

blurbs は次のように定義されています。

```
create table blurbs
(au_id  id not null,
copy text null)
```

プライマリ・キーは au_id です。

```
sp_primarykey blurbs, au_id
```

au_id カラムは、authors に対する外部キーです。

```
sp_foreignkey blurbs, authors, au_id
```

au_pix テーブル

pubs2 データベースの author_pix テーブルの内容は、作家の写真です。

au_pix は次のように定義されています。

```
create table au_pix
(au_id char(11) not null,
pic image null,
format_type char(11) null,
bytesize int null,
pixwidth_hor char(14) null,
pixwidth_vert char(14) null)
```

プライマリ・キーは au_id です。

```
sp_primarykey au_pix, au_id
```

au_id カラムは、authors に対する外部キーです。

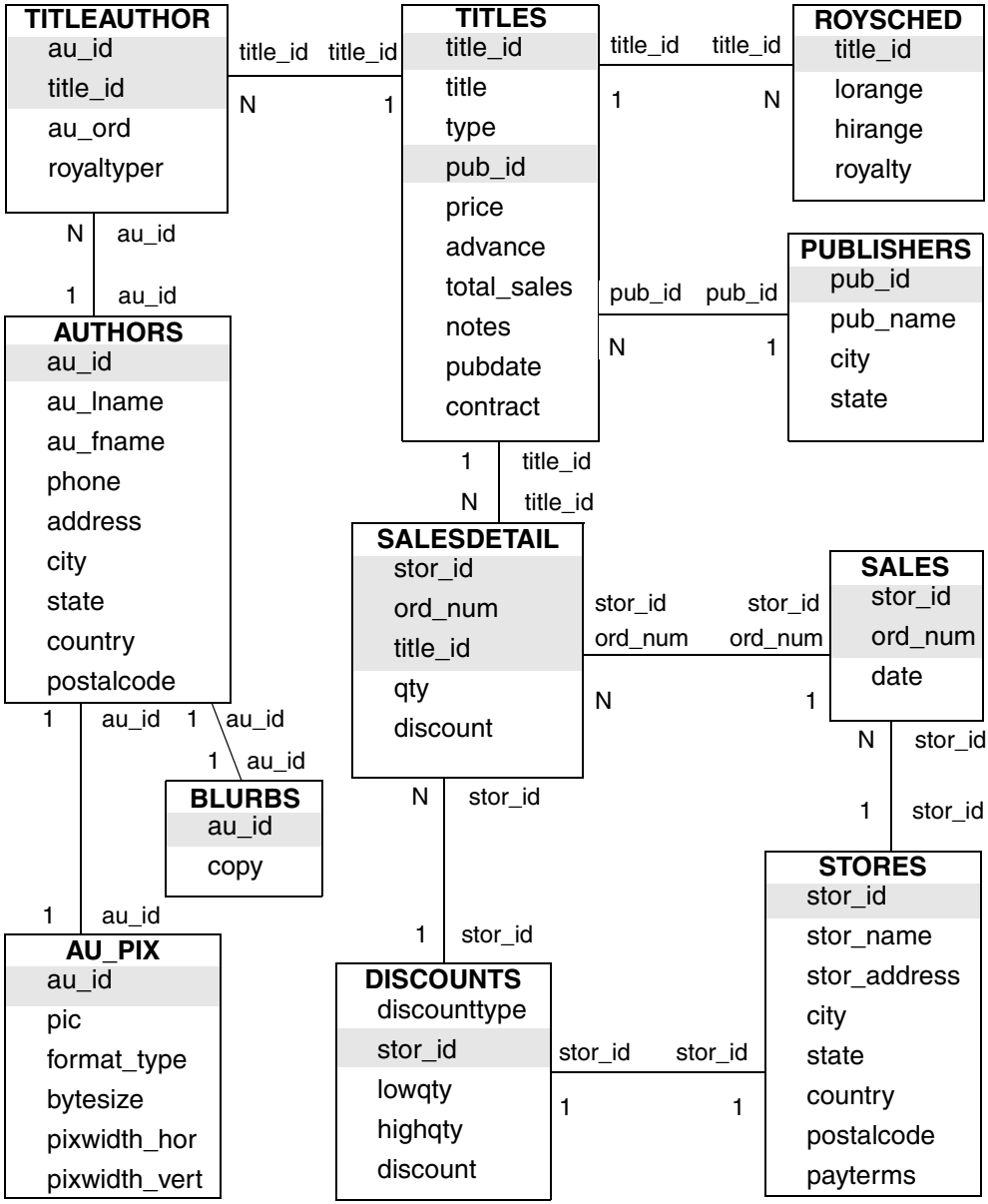
```
sp_foreignkey au_pix, authors, au_id
```

pic カラムはバイナリ・データを格納します。image データ (写真は 6 枚で、PICT、TIF、Sunraster ファイル・フォーマットが 2 枚ずつ) は非常にサイズが大きいので、image データ型を使用するかテストする場合にだけ、installpix2 スクリプトを実行してください。image データは、Sybase が image データを格納する方法を示すために提供されます。Sybase では image データを表示するツールを用意していません。イメージをデータベースから抽出したら、適切なグラフィックス・ツールを使用してそのイメージを表示してください。

pubs2 データベースの関係図

図 A-1 は、pubs2 データベース内のテーブルとテーブル間の関係の一部を示しています。

図 A-1: pubs2 データベースの関係図



付 録　B

pubs3 データベース

この付録では、サンプル・データベース pubs3 について説明します。このサンプル・データベースには、publishers、authors、titles、titleauthor、salesdetail、sales、stores、store_employees、roysched、discounts、および blurbs の各テーブルが含まれています。

また、各テーブルを作成するために使用するプライマリ・キーと外部キー、ルール、デフォルト、ビュー、トリガ、ストアド・プロシージャも含まれます。

pubs3 データベースの関係図は図 B-1 (719 ページ) にあります。

pubs3 のインストールについては、使用しているプラットフォームの『インストール・ガイド』を参照してください。

create やデータ修正文を使用してサンプル・データベースを変更するには、システム管理者から追加のパーミッションを取得する必要があります。サンプル・データベースを変更する場合は、次のユーザが使用するときのために、データベースを元の状態に戻すことをおすすめします。サンプル・データベースのリストアにヘルプが必要な場合は、システム管理者に連絡してください。

pubs3 データベース内のテーブル

pubs3 データベースの各テーブル内の各カラム・ヘッダは、カラム名、そのデータ型 (ユーザ定義データ型も含む)、その null または not null のステータス、参照整合性の使用方法を示しています。また、カラム・ヘッダは、カラムに影響を与えるデフォルト、ルール、トリガ、インデックスも示しています。

publishers テーブル

publishers テーブルの内容は、出版社の ID、名前、都市、州です。

publishers は次のように定義されています。

```
create table publishers
 (pub_id char (4) not null,
pub_name varchar(40) not null,
city varchar(20) null,
state char(2) null,
unique nonclustered (pub_id))
```

pub_idrule ルールは次のように定義されています。

```
create rule pub_idrule
as @pub_id in
("1389", "0736", "0877", "1622", "1756")
or @pub_id like "99[0-9][0-9]"
```

authors テーブル

authors テーブルの内容は、作家の名前、電話番号、その他の情報です。

authors は次のように定義されています。

```
create table authors
(au_id id not null,
au_lname varchar(40) not null,
au_fname varchar(20) not null,
phone char(12) not null,
address varchar(40) null,
city varchar(20) null,
state char(2) null,
country varchar(12) null,
postalcode char(10) null,
unique nonclustered (au_id))
```

au_lname カラムと au_fname カラムのノンクラスタード・インデックスは次のように定義されています。

```
create nonclustered index aunmind
on authors (au_lname, au_fname)
```

phone カラムには以下のデフォルトが使用されています。

```
create default phonedflt as "UNKNOWN"
sp_bindefault phonedft, "authors.phone"
```

次のビューは authors を使用しています。

```
create view titleview
as
select title, au_ord, au_lname,
price, num_sold, pub_id
from authors, titles, titleauthor
where authors.au_id = titleauthor.au_id
and titles.title_id = titleauthor.title_id
```

titles テーブル

titles テーブルの内容は、タイトルの名前、タイトル ID、種類、その他の情報です。

titles は次のように定義されています。

```
create table titles
(title_id tid not null,
title varchar(80) not null,
type char(12) not null,
pub_id char(4) null
    references publishers(pub_id),
price money null,
advance numeric(12,2) null,
num_sold int null,
notes varchar(200) null,
pubdate datetime not null,
contract bit not null,
unique nonclustered (title_id))
```

title カラムのノンクラスタード・インデックスは次のように定義されています。

```
create nonclustered index titleind
on titles(title)
```

title_idrule は次のように定義されています。

```
create rule title_idrule
as
@title_id like "BU[0-9][0-9][0-9][0-9]" or
@title_id like "[MT]C[0-9][0-9][0-9][0-9]" or
@title_id like "P[SC][0-9][0-9][0-9][0-9]" or
@title_id like "[A-Z][A-Z]xxxx" or
@title_id like "[A-Z][A-Z]yyyy"
```

type カラムには以下のデフォルトが使用されています。

```
create default typedflt as "UNDECIDED"
sp_bindefault typedflt, "titles.type"
```

pubdate カラムには以下のデフォルトが設定されています。

```
create default datedflt as getdate()
sp_bindefault datedflt, "titles.pubdate"
```

titles は次のトリガを使用します。

```
create trigger deltitle
on titles
for delete
as
if (select count(*) from deleted, salesdetail
where salesdetail.title_id = deleted.title_id) >0
begin
   rollback transaction
```

```
    print "You can't delete a title with sales."
end
```

次のビューは titles を使用しています。

```
create view titleview
as
select title, au_ord, au_lname,
price, num_sold, pub_id
from authors, titles, titleauthor
where authors.au_id = titleauthor.au_id
and titles.title_id = titleauthor.title_id
```

titleauthor テーブル

titleauthor テーブルの内容は、タイトルと作家の ID、印税のパーセンテージ、その他の情報です。

titleauthor は次のように定義されています。

```
create table titleauthor
(au_id id not null
    references authors(au_id),
title_id tid not null
    references titles(title_id),
au_ord tinyint null,
royaltyper int null)
```

au_id カラムのノンクラスタード・インデックスは次のように定義されています。

```
create nonclustered index auidind
on titleauthor(au_id)
```

title_id のノンクラスタード・インデックスは次のように定義されています。

```
create nonclustered index titleidind
on titleauthor(title_id)
```

次のビューは titleauthor を使用しています。

```
create view titleview
as
select title, au_ord, au_lname,
price, num_sold, pub_id
from authors, titles, titleauthor
where authors.au_id = titleauthor.au_id
and titles.title_id = titleauthor.title_id
```

このプロシージャは titleauthor を使用しています。

```
create procedure byroyalty @percentage int
as
select au_id from titleauthor
where titleauthor.royaltyper = @percentage
```

salesdetail テーブル

salesdetail テーブルの内容は、書店 ID、注文番号、およびその他の販売に関する詳細です。

salesdetail は次のように定義されています。

```
create table salesdetail
(stor_id char(4) not null
    references sales(stor_id),
ord_num numeric(6,0)
    references sales(ord_num),
title_id tid not null
    references titles(title_id),
qty smallint not null,
discount float not null)
```

title_id カラムのノンクラスタード・インデックスは次のように定義されています。

```
create nonclustered index titleidind
on salesdetail (title_id)
```

stor_id カラムのノンクラスタード・インデックスは次のように定義されています。

```
create nonclustered index salesdetailind
on salesdetail (stor_id)
```

title_idrule ルールは次のように定義されています。

```
create rule title_idrule
as
@title_id like "BU[0-9][0-9][0-9][0-9]" or
@title_id like "[MT]C[0-9][0-9][0-9][0-9]" or
@title_id like "P[SC][0-9][0-9][0-9][0-9]" or
@title_id like "[A-Z][A-Z]xxxx" or
@title_id like "[A-Z][A-Z]yyyy"
```

salesdetail は次のトリガを使用します。

```
create trigger totalsales_trig on salesdetail
    for insert, update, delete
as
/* Save processing:  return if there are no rows affected */
if @@rowcount = 0
    begin
      return
end
/* add all the new values */
/* use isnull:  a null value in the titles table means
**              "no sales yet" not "sales unknown"
*/
update titles
    set num_sold = isnull(num_sold, 0) + (select sum(qty)
```

```
    from inserted
    where titles.title_id = inserted.title_id)
    where title_id in (select title_id from inserted)
/* remove all values being deleted or updated */
update titles
    set num_sold = isnull(num_sold, 0) - (select sum(qty)
    from deleted
    where titles.title_id = deleted.title_id)
    where title_id in (select title_id from deleted)
```

sales テーブル

sales テーブルの内容は、販売書店 ID、注文番号、販売日です。

sales は次のように定義されています。

```
create table sales
(stor_id char(4) not null
     references stores(stor_id),
ord_num numeric(6,0) identity,
date datetime not null,
unique nonclustered (ord_num))
```

stores テーブル

stores テーブルの内容は、書店 ID、書店名、およびその他の書店に関する情報です。

stores は次のように定義されています。

```
create table stores
(stor_id char(4) not null,
stor_name varchar(40) not null,
stor_address varchar(40) null,
city varchar(20) null,
state char(2) null,
country varchar(12) null,
postalcode char(10) null,
payterms varchar(12) null,
unique nonclustered (stor_id))
```

store_employees テーブル

store_employees テーブルの内容は、書店、従業員、マネージャ ID、およびその他の書店従業員に関する情報です。

store_employees は次のように定義されています。

```
create table store_employees
(stor_id char(4) null
     references stores(stor_id),
emp_id id not null,
mgr_id id null
     references store_employees(emp_id),
emp_lname varchar(40) not null,
emp_fname varchar(20) not null,
phone char(12) null,
address varchar(40) null,
city varchar(20) null,
state char(2) null,
country varchar(12) null,
postalcode varchar(10) null,
unique nonclustered (emp_id))
```

roysched テーブル

roysched テーブルの内容は、タイトル ID、印税のパーセンテージ、およびその他のタイトル印税に関する情報です。

roysched は次のように定義されています。

```
create table roysched
title_id tid not null
    references titles(title_id),
lorange int null,
hirange int null,
royalty int null)
```

title_id カラムのノンクラスタード・インデックスは次のように定義されています。

```
create nonclustered index titleidind
on roysched (title_id)
```

discounts テーブル

discount テーブルの内容は、値引きの種類、書店 ID、冊数、および値引き率です。

discounts は次のように定義されています。

```
create table discounts
(discounttype varchar(40) not null,
stor_id char(4) null
     references stores(stor_id),
lowqty smallint null,
highqty smallint null,
discount float not null)
```

blurbs テーブル

pubs3 データベースの blurbs テーブルの内容は、作家 ID と本の広告です。

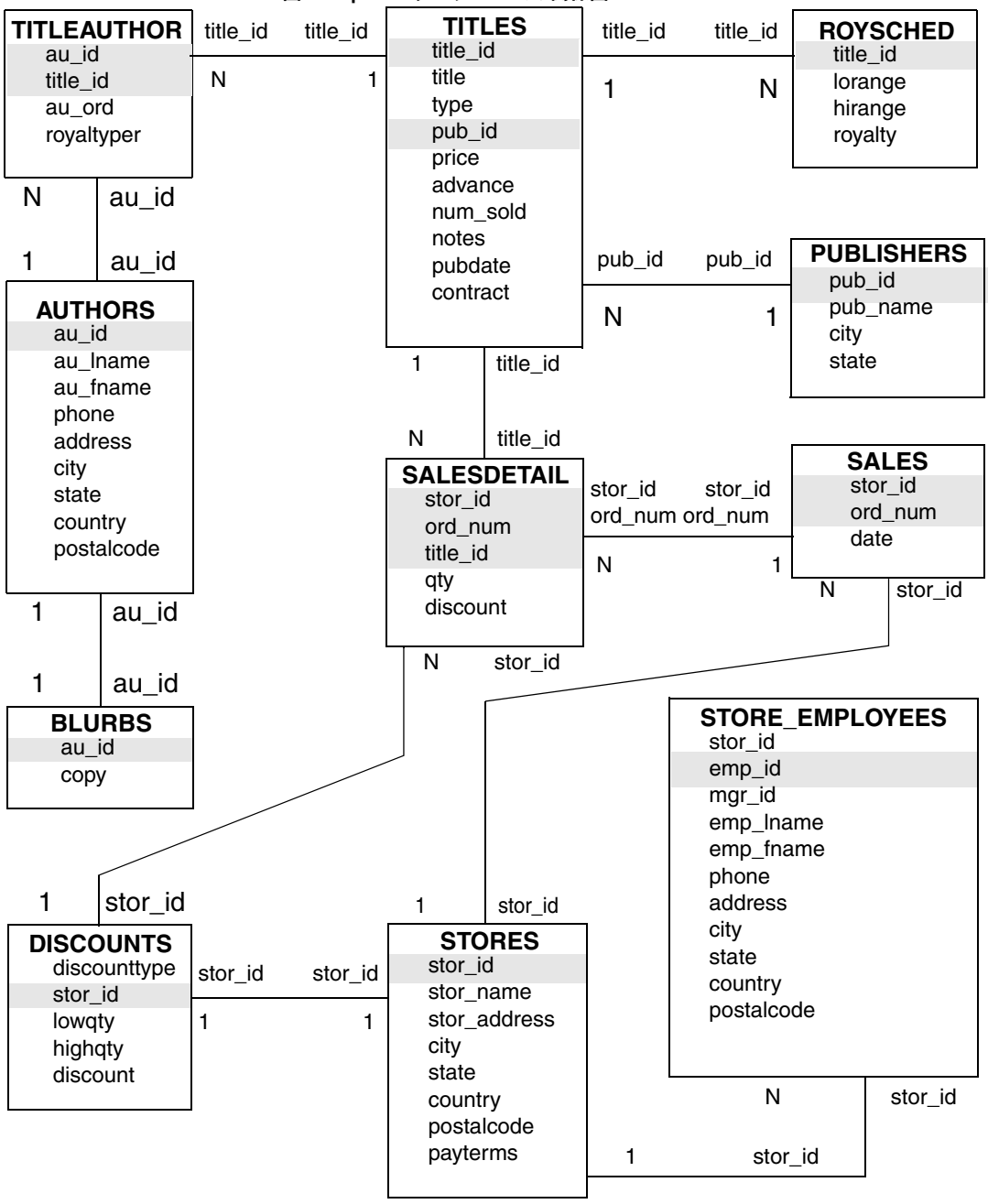
blurbs は次のように定義されています。

```
create table blurbs
(au_id  id not null
    references authors(au_id),
copy text null)
```

pubs3 データベースの関係図

図 B-1 は、pubs3 データベース内のテーブルとテーブル間の関係の一部を示しています。

図 B-1: pubs3 データベースの関係図



索引

記号

数字

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

T

U

V

W

X

Y

あ

い

う

え

お

か

き

く

け

こ

し

す

せ

そ

た

ち

つ

て

と

な

に

ね

の

は

ひ

ふ

や

ゆ

よ

ら

り

る