Reference Manual: Building Blocks

Adaptive Server® Enterprise

15.5
DOCUMENT ID: DC36271-01-1550-01

LAST REVISED: November 2009

Copyright © 2009 by Sybase, Inc. All rights reserved.

This publication pertains to Sybase software and to any subsequent release until otherwise indicated in new editions or technical notes. Information in this document is subject to change without notice. The software described herein is furnished under a license agreement, and it may be used or copied only in accordance with the terms of that agreement.

To order additional documents, U.S. and Canadian customers should call Customer Fulfillment at (800) 685-8225, fax (617) 229-9845.

Customers in other countries with a U.S. license agreement may contact Customer Fulfillment via the above fax number. All other international customers should contact their Sybase subsidiary or local distributor. Upgrades are provided only at regularly scheduled software release dates. No part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without the prior written permission of Sybase, Inc.

Sybase trademarks can be viewed at the Sybase trademarks page at http://www.sybase.com/detail?id=1011207. Sybase and the marks listed are trademarks of Sybase, Inc. ® indicates registration in the United States of America.

Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Unicode and the Unicode Logo are registered trademarks of Unicode, Inc.

IBM and Tivoli are registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

All other company and product names mentioned may be trademarks of the respective companies with which they are associated.

Use, duplication, or disclosure by the government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.

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

About This Book ......................................................................................................................... xiii

CHAPTER 1  System and User-Defined Datatypes ................................................................. 1
Datatype categories .................................................................................................................... 1
Range and storage size .............................................................................................................. 2
Datatypes of columns, variables, or parameters ................................................................. 4
  Declaring the datatype for a column in a table ................................................................. 5
  Declaring the datatype for a local variable in a batch or procedure .................................. 5
  Declaring the datatype for a parameter in a stored procedure ........................................... 5
Determining the datatype of a literal ..................................................................................... 6
Datatypes of mixed-mode expressions ................................................................................... 7
Determining the datatype hierarchy ....................................................................................... 7
Determining precision and scale ............................................................................................. 9
Datatype conversions ............................................................................................................ 9
  Automatic conversion of fixed-length NULL columns ...................................................... 10
  Handling overflow and truncation errors ....................................................................... 10
Standards and compliance .................................................................................................... 11
Exact numeric datatypes ....................................................................................................... 12
  Integer types ..................................................................................................................... 13
  Decimal datatypes ............................................................................................................ 14
  Standards and compliance ............................................................................................... 15
Approximate numeric datatypes ......................................................................................... 16
  Understanding approximate numeric datatypes ............................................................... 16
  Range, precision, and storage size ................................................................................... 17
  Entering approximate numeric data ............................................................................... 17
  NaN and Inf values ......................................................................................................... 18
  Standards and compliance ............................................................................................... 18
Money datatypes .................................................................................................................... 18
  Accuracy ........................................................................................................................... 18
  Range and storage size ..................................................................................................... 18
  Entering monetary values ............................................................................................... 19
  Standards and compliance ............................................................................................... 19
Timestamp datatype ............................................................................................................. 19
## Contents

Aggregate functions and NULL values .................................................. 55
Vector and scalar aggregates ................................................................. 55
Aggregate functions as row aggregates .................................................. 57
Statistical aggregate functions ................................................................. 60
Standard deviation and variance ............................................................. 60
Statistical aggregates ............................................................................ 61
Datatype conversion functions ................................................................. 63
Converting character data to a noncharacter type .................................... 67
Converting from one character type to another ....................................... 67
Converting numbers to a character type .................................................. 68
Rounding during conversion to and from money types ............................. 68
Converting date and time information ..................................................... 69
Converting between numeric types .......................................................... 69
Arithmetic overflow and divide-by-zero errors ....................................... 70
Conversions between binary and integer types ........................................ 71
Converting between binary and numeric or decimal types ........................ 72
Converting image columns to binary types ............................................. 72
Converting other types to bit .................................................................. 72
Converting NULL value ........................................................................... 73
Date functions .......................................................................................... 73
Date parts .................................................................................................. 73
Mathematical functions ........................................................................... 74
Security functions ..................................................................................... 75
String functions ........................................................................................ 76
Limits on string functions ......................................................................... 77
System functions ...................................................................................... 77
Text, unitext, and image columns .......................................................... 77
Text and image functions ......................................................................... 78
User-defined SQL functions ..................................................................... 78
abs ............................................................................................................. 80
acos ........................................................................................................... 81
ascii .......................................................................................................... 82
asehostname ............................................................................................ 83
asin ............................................................................................................. 84
atan ........................................................................................................... 85
atn2 ............................................................................................................ 86
avg .............................................................................................................. 87
audit_event_name ..................................................................................... 89
authmech ................................................................................................. 91
biginttohex ............................................................................................... 92
bintostr ..................................................................................................... 93
cache_usage ............................................................................................ 95
case .......................................................................................................... 96
cast ............................................................................................................ 99
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ceiling</td>
<td>102</td>
</tr>
<tr>
<td>char</td>
<td>104</td>
</tr>
<tr>
<td>char_length</td>
<td>106</td>
</tr>
<tr>
<td>charindex</td>
<td>108</td>
</tr>
<tr>
<td>coalesce</td>
<td>109</td>
</tr>
<tr>
<td>col_length</td>
<td>111</td>
</tr>
<tr>
<td>col_name</td>
<td>112</td>
</tr>
<tr>
<td>compare</td>
<td>113</td>
</tr>
<tr>
<td>convert</td>
<td>118</td>
</tr>
<tr>
<td>cos</td>
<td>124</td>
</tr>
<tr>
<td>cot</td>
<td>125</td>
</tr>
<tr>
<td>count</td>
<td>126</td>
</tr>
<tr>
<td>count_big</td>
<td>128</td>
</tr>
<tr>
<td>current_bigdatetime</td>
<td>130</td>
</tr>
<tr>
<td>current_bigtime</td>
<td>131</td>
</tr>
<tr>
<td>current_date</td>
<td>132</td>
</tr>
<tr>
<td>current_time</td>
<td>133</td>
</tr>
<tr>
<td>curunreservedpgs</td>
<td>134</td>
</tr>
<tr>
<td>data_pages</td>
<td>136</td>
</tr>
<tr>
<td>datachange</td>
<td>138</td>
</tr>
<tr>
<td>datalength</td>
<td>140</td>
</tr>
<tr>
<td>dateadd</td>
<td>141</td>
</tr>
<tr>
<td>datediff</td>
<td>144</td>
</tr>
<tr>
<td>datename</td>
<td>148</td>
</tr>
<tr>
<td>datepart</td>
<td>150</td>
</tr>
<tr>
<td>day</td>
<td>155</td>
</tr>
<tr>
<td>db_attr</td>
<td>156</td>
</tr>
<tr>
<td>db_id</td>
<td>158</td>
</tr>
<tr>
<td>db_instanceid</td>
<td>159</td>
</tr>
<tr>
<td>db_name</td>
<td>160</td>
</tr>
<tr>
<td>db_recovery_status</td>
<td>161</td>
</tr>
<tr>
<td>degrees</td>
<td>162</td>
</tr>
<tr>
<td>derived_stat</td>
<td>163</td>
</tr>
<tr>
<td>difference</td>
<td>168</td>
</tr>
<tr>
<td>exp</td>
<td>169</td>
</tr>
<tr>
<td>floor</td>
<td>170</td>
</tr>
<tr>
<td>get_appcontext</td>
<td>172</td>
</tr>
<tr>
<td>getdate</td>
<td>173</td>
</tr>
<tr>
<td>getutcdate</td>
<td>174</td>
</tr>
<tr>
<td>has_role</td>
<td>175</td>
</tr>
<tr>
<td>hash</td>
<td>177</td>
</tr>
<tr>
<td>hashbytes</td>
<td>179</td>
</tr>
<tr>
<td>hextobigint</td>
<td>181</td>
</tr>
<tr>
<td>hextoint</td>
<td>182</td>
</tr>
</tbody>
</table>
host_id.......................................................................................... 183
host_name ................................................................................... 184
instance_id ................................................................................... 185
identity_burn_max................................................................. 186
index_col ............................................................................... 187
index_colorder ......................................................................... 188
index_name............................................................................... 189
inttohex ................................................................................. 190
isdate ..................................................................................... 191
isnumeric ............................................................................... 192
is_quiesced ........................................................................ 193
is_sec_service_on ............................................................. 195
isnull .................................................................................. 196
isnumeric ............................................................................... 197
instance_name ........................................................................ 198
lc_id ...................................................................................... 199
lc_name .................................................................................. 200
lct_admin ............................................................................... 201
left ..................................................................................... 204
len ........................................................................................ 205
license_enabled ..................................................................... 206
list_appcontext ....................................................................... 207
lockscheme ............................................................................. 208
log ...................................................................................... 209
log10 ................................................................................... 210
lower ................................................................................... 211
ltrim ..................................................................................... 212
max ...................................................................................... 213
min ....................................................................................... 215
month .................................................................................. 216
mut_excl_roles ...................................................................... 217
newid ..................................................................................... 218
next_identity ........................................................................ 220
nullif ..................................................................................... 221
object_attr ............................................................................ 223
object_id ............................................................................... 227
object_name ........................................................................... 228
object_owner_id ..................................................................... 229
pagesize ............................................................................... 230
partition_id .......................................................................... 232
partition_name ....................................................................... 233
partition_object_id ................................................................... 234
patindex .................................................................................. 235
pi ......................................................................................... 238
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>power</td>
<td>239</td>
</tr>
<tr>
<td>proc_role</td>
<td>240</td>
</tr>
<tr>
<td>pssinfo</td>
<td>242</td>
</tr>
<tr>
<td>radians</td>
<td>243</td>
</tr>
<tr>
<td>rand</td>
<td>244</td>
</tr>
<tr>
<td>rand2</td>
<td>245</td>
</tr>
<tr>
<td>replicate</td>
<td>246</td>
</tr>
<tr>
<td>reserve_identity</td>
<td>247</td>
</tr>
<tr>
<td>reserved_pages</td>
<td>250</td>
</tr>
<tr>
<td>reverse</td>
<td>254</td>
</tr>
<tr>
<td>right</td>
<td>255</td>
</tr>
<tr>
<td>rm_appcontext</td>
<td>257</td>
</tr>
<tr>
<td>role_contain</td>
<td>258</td>
</tr>
<tr>
<td>role_id</td>
<td>259</td>
</tr>
<tr>
<td>role_name</td>
<td>260</td>
</tr>
<tr>
<td>round</td>
<td>261</td>
</tr>
<tr>
<td>row_count</td>
<td>263</td>
</tr>
<tr>
<td>rtrim</td>
<td>264</td>
</tr>
<tr>
<td>sdc_intempdbconfig</td>
<td>265</td>
</tr>
<tr>
<td>set_appcontext</td>
<td>266</td>
</tr>
<tr>
<td>show_role</td>
<td>268</td>
</tr>
<tr>
<td>show_sec_services</td>
<td>269</td>
</tr>
<tr>
<td>sign</td>
<td>270</td>
</tr>
<tr>
<td>sin</td>
<td>271</td>
</tr>
<tr>
<td>sortkey</td>
<td>272</td>
</tr>
<tr>
<td>soundex</td>
<td>277</td>
</tr>
<tr>
<td>space</td>
<td>278</td>
</tr>
<tr>
<td>spid_instance_id</td>
<td>279</td>
</tr>
<tr>
<td>square</td>
<td>280</td>
</tr>
<tr>
<td>sqrt</td>
<td>281</td>
</tr>
<tr>
<td>stddev</td>
<td>282</td>
</tr>
<tr>
<td>stdev</td>
<td>283</td>
</tr>
<tr>
<td>stdevp</td>
<td>284</td>
</tr>
<tr>
<td>stddev_pop</td>
<td>285</td>
</tr>
<tr>
<td>stddev_samp</td>
<td>287</td>
</tr>
<tr>
<td>str</td>
<td>289</td>
</tr>
<tr>
<td>str_replace</td>
<td>291</td>
</tr>
<tr>
<td>strtonin</td>
<td>293</td>
</tr>
<tr>
<td>stuff</td>
<td>295</td>
</tr>
<tr>
<td>substring</td>
<td>297</td>
</tr>
<tr>
<td>sum</td>
<td>299</td>
</tr>
<tr>
<td>user_id</td>
<td>301</td>
</tr>
<tr>
<td>user_name</td>
<td>302</td>
</tr>
<tr>
<td>syb_quit</td>
<td>303</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>syb_sendmsg</td>
<td>304</td>
</tr>
<tr>
<td>sys_tempdbid</td>
<td>305</td>
</tr>
<tr>
<td>tan</td>
<td>306</td>
</tr>
<tr>
<td>tempdb_id</td>
<td>307</td>
</tr>
<tr>
<td>textptr</td>
<td>308</td>
</tr>
<tr>
<td>textvalid</td>
<td>309</td>
</tr>
<tr>
<td>to_unichar</td>
<td>310</td>
</tr>
<tr>
<td>tran_dumpable_status</td>
<td>311</td>
</tr>
<tr>
<td>tsequal</td>
<td>313</td>
</tr>
<tr>
<td>uhighsurr</td>
<td>315</td>
</tr>
<tr>
<td>ulowsurr</td>
<td>316</td>
</tr>
<tr>
<td>upper</td>
<td>317</td>
</tr>
<tr>
<td>uscalar</td>
<td>318</td>
</tr>
<tr>
<td>used_pages</td>
<td>319</td>
</tr>
<tr>
<td>user</td>
<td>321</td>
</tr>
<tr>
<td>user_id</td>
<td>322</td>
</tr>
<tr>
<td>user_name</td>
<td>323</td>
</tr>
<tr>
<td>valid_name</td>
<td>324</td>
</tr>
<tr>
<td>valid_user</td>
<td>325</td>
</tr>
<tr>
<td>var</td>
<td>326</td>
</tr>
<tr>
<td>var_pop</td>
<td>327</td>
</tr>
<tr>
<td>var_samp</td>
<td>329</td>
</tr>
<tr>
<td>variance</td>
<td>330</td>
</tr>
<tr>
<td>varp</td>
<td>331</td>
</tr>
<tr>
<td>workload_metric</td>
<td>332</td>
</tr>
<tr>
<td>xa_bqual</td>
<td>333</td>
</tr>
<tr>
<td>xa_grtid</td>
<td>335</td>
</tr>
<tr>
<td>xmltable</td>
<td>337</td>
</tr>
<tr>
<td>year</td>
<td>350</td>
</tr>
</tbody>
</table>

**CHAPTER 3**

Global Variables ......................................................... 351
Adaptive Server global variables.................................... 351
Using global variables in a clustered environment............. 358

**CHAPTER 4**

Expressions, Identifiers, and Wildcard Characters .............. 359
Expressions........................................................................ 359
Size of expressions .................................................... 360
Arithmetic and character expressions .............................. 360
Relational and logical expressions .................................. 360
Operator precedence .................................................... 361
Arithmetic operators .................................................... 361
Bitwise operators ......................................................... 362
String concatenation operator .............................. 363
Contents

Transaction rollbacks ........................................................................... 394
with check option violation................................................................. 394

Index ................................................................................................... 397
The *Adaptive Server Reference Manual* includes four guides to Sybase®
Adaptive Server.® Enterprise and the Transact-SQL.® language:

- **Building Blocks** describes the “parts” of Transact-SQL: datatypes,
built-in functions, global variables, expressions and identifiers,
reserved words, and SQLSTATE errors. Before you can use
Transact-SQL successfully, you must understand what these building
blocks do and how they affect the results of Transact-SQL statements.

- **Commands** provides reference information about the Transact-SQL
commands, which you use to create statements.

- **Procedures** provides reference information about system procedures,
catalog stored procedures, extended stored procedures, and dbcc
stored procedures. All procedures are created using Transact-SQL
statements.

- **Tables** provides reference information about the system tables, which
store information about your server, databases, users, and other
details of your server. It also provides information about the tables in
the dbccdb and dbccalt databases.

**Audience**

The *Adaptive Server Reference Manual* is intended as a reference tool for
Transact-SQL users of all levels.

**How to use this book**

- Chapter 1, “System and User-Defined Datatypes,” describes the
system and user-defined datatypes that are supplied with Adaptive
Server and indicates how to use them to create user-defined
datatypes.

- Chapter 2, “Transact-SQL Functions,” lists the Adaptive Server
functions in a table that provides the name and a brief description.

- Chapter 3, “Global Variables,” lists the system-defined variables for
Adaptive Server in a table that provides the name and a brief
description of the returned status.

- Chapter 4, “Expressions, Identifiers, and Wildcard Characters,”
which provides information about using the Transact-SQL language.
Chapter 5, “Reserved Words,” provides information about the Transact-SQL and ANSI SQL keywords.

Chapter 6, “SQLSTATE Codes and Messages,” contains information about Adaptive Server SQLSTATE status codes and the associated messages.

Related documents

The Adaptive Server® Enterprise documentation set consists of:

- The release bulletin for your platform – contains last-minute information that was too late to be included in the books.
  A more recent version of the release bulletin may be available. To check for critical product or document information that was added after the release of the product CD, use the Sybase® Product Manuals Web site.
- The installation guide for your platform – describes installation, upgrading, and some configuration procedures for all Adaptive Server and related Sybase products.
- New Feature Summary – describes the new features in Adaptive Server, the system changes added to support those features, and changes that may affect your existing applications.
- Active Messaging Users Guide – describes how to use the Active Messaging feature to capture transactions (data changes) in an Adaptive Server Enterprise database, and deliver them as events to external applications in real time.
- Component Integration Services Users Guide – explains how to use Component Integration Services to connect remote Sybase and non-Sybase databases.
- The Configuration Guide for your platform – provides instructions for performing specific configuration tasks.
- Glossary – defines technical terms used in the Adaptive Server documentation.
- Java in Adaptive Server Enterprise – describes how to install and use Java classes as datatypes, functions, and stored procedures in the Adaptive Server database.
- Job Scheduler Users Guide – provides instructions on how to install and configure, and create and schedule jobs on a local or remote Adaptive Server using the command line or a graphical user interface (GUI).
• Migration Technology Guide – describes strategies and tools for migrating to a different version of Adaptive Server.
• Monitor Client Library Programmers Guide – describes how to write Monitor Client Library applications that access Adaptive Server performance data.
• Monitoring Tables Diagram – illustrates monitor tables and their entity relationships in a poster format. Full-size available only in print version; a compact version is available in PDF format.
• Performance and Tuning Series – is a series of books that explain how to tune Adaptive Server for maximum performance:
  • Basics – contains the basics for understanding and investigating performance questions in Adaptive Server.
  • Improving Performance with Statistical Analysis – describes how Adaptive Server stores and displays statistics, and how to use the set statistics command to analyze server statistics.
  • Locking and Concurrency Control – describes how to use locking schemes to improve performance, and how to select indexes to minimize concurrency.
  • Monitoring Adaptive Server with sp_sysmon – discusses how to use sp_sysmon to monitor performance.
  • Monitoring Tables – describes how to query Adaptive Server monitoring tables for statistical and diagnostic information.
  • Physical Database Tuning – describes how to manage physical data placement, space allocated for data, and the temporary databases.
  • Query Processing and Abstract Plans – explains how the optimizer processes queries, and how to use abstract plans to change some of the optimizer plans.
• Quick Reference Guide – provides a comprehensive listing of the names and syntax for commands, functions, system procedures, extended system procedures, datatypes, and utilities in a pocket-sized book (regular size when viewed in PDF format).
• Reference Manual – is a series of books that contains detailed Transact-SQL® information:
• **Building Blocks** – discusses datatypes, functions, global variables, expressions, identifiers and wildcards, and reserved words.

• **Commands** – documents commands.

• **Procedures** – describes system procedures, catalog stored procedures, system extended stored procedures, and dbcc stored procedures.

• **Tables** – discusses system tables, monitor tables, and dbcc tables.

• **System Administration Guide** –
  
  • **Volume 1** – provides an introduction to the basics of system administration, including a description of configuration parameters, resource issues, character sets, sort orders, and instructions for diagnosing system problems. The second part of **Volume 1** is an in-depth discussion about security administration.

  • **Volume 2** – includes instructions and guidelines for managing physical resources, mirroring devices, configuring memory and data caches, managing multiprocessor servers and user databases, mounting and unmounting databases, creating and using segments, using the `reorg` command, and checking database consistency. The second half of **Volume 2** describes how to back up and restore system and user databases.

• **System Tables Diagram** – illustrates system tables and their entity relationships in a poster format. Full-size available only in print version; a compact version is available in PDF format.

• **Transact-SQL Users Guide** – documents Transact-SQL, the Sybase-enhanced version of the relational database language. This guide serves as a textbook for beginning users of the database management system, and also contains detailed descriptions of the `pubs2` and `pubs3` sample databases.

• **Troubleshooting: Error Messages Advanced Resolutions** – contains troubleshooting procedures for problems you may encounter. The problems discussed here are the ones the Sybase Technical Support staff hear about most often.

• **Encrypted Columns Users Guide** – describes how to configure and use encrypted columns with Adaptive Server.

• **In-Memory Database Users Guide** – describes how to configure and use in-memory databases.
• *Using Adaptive Server Distributed Transaction Management Features* – explains how to configure, use, and troubleshoot Adaptive Server DTM features in distributed transaction processing environments.

• *Using Backup Server with IBM® Tivoli® Storage Manager* – describes how to set up and use the IBM Tivoli Storage Manager to create Adaptive Server backups.

• *Using Sybase Failover in a High Availability System* – provides instructions for using Sybase Failover to configure an Adaptive Server as a companion server in a high availability system.

• *Unified Agent and Agent Management Console* – describes the Unified Agent, which provides runtime services to manage, monitor, and control distributed Sybase resources.

• *Utility Guide* – documents the Adaptive Server utility programs, such as `isql` and `bcp`, which are executed at the operating system level.

• *Web Services Users Guide* – explains how to configure, use, and troubleshoot Web services for Adaptive Server.

• *XA Interface Integration Guide for CICS, Encina, and TUXEDO* – provides instructions for using the Sybase DTM XA interface with X/Open XA transaction managers.

• *XML Services in Adaptive Server Enterprise* – describes the Sybase native XML processor and the Sybase Java-based XML support, introduces XML in the database, and documents the query and mapping functions that are available in XML services.

**Other sources of information**

Use the Sybase Getting Started CD, the SyBooks™ CD, and the Sybase Product Manuals Web site to learn more about your product:

• The Getting Started CD contains release bulletins and installation guides in PDF format, and may also contain other documents or updated information not included on the SyBooks CD. It is included with your software. To read or print documents on the Getting Started CD, you need Adobe Acrobat Reader, which you can download at no charge from the Adobe Web site using a link provided on the CD.

• The SyBooks CD contains product manuals and is included with your software. The Eclipse-based SyBooks browser allows you to access the manuals in an easy-to-use, HTML-based format. Some documentation may be provided in PDF format, which you can access through the PDF directory on the SyBooks CD. To read or print the PDF files, you need Adobe Acrobat Reader.
Refer to the *SyBooks Installation Guide* on the Getting Started CD, or the *README.txt* file on the SyBooks CD for instructions on installing and starting SyBooks.

- The Sybase Product Manuals Web site is an online version of the SyBooks CD that you can access using a standard Web browser. In addition to product manuals, you will find links to EBFs/Maintenance, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

  To access the Sybase Product Manuals Web site, go to Product Manuals at http://www.sybase.com/support/manuals/.

**Sybase certifications on the Web**

Technical documentation at the Sybase Web site is updated frequently.

- **Finding the latest information on product certifications**
  2. Click Certification Report.
  3. In the Certification Report filter select a product, platform, and timeframe and then click Go.
  4. Click a Certification Report title to display the report.

- **Finding the latest information on component certifications**
  2. Either select the product family and product under Search by Base Product; or select the platform and product under Search by Platform.
  3. Select Search to display the availability and certification report for the selection.

- **Creating a personalized view of the Sybase Web site (including support pages)**
  Set up a MySybase profile. MySybase is a free service that allows you to create a personalized view of Sybase Web pages.
  2. Click MySybase and create a MySybase profile.
Finding the latest information on EBFs and software maintenance

2. Select EBFs/Maintenance. If prompted, enter your MySybase user name and password.
3. Select a product.
4. Specify a time frame and click Go. A list of EBF/Maintenance releases is displayed.
   Padlock icons indicate that you do not have download authorization for certain EBF/Maintenance releases because you are not registered as a Technical Support Contact. If you have not registered, but have valid information provided by your Sybase representative or through your support contract, click Edit Roles to add the “Technical Support Contact” role to your MySybase profile.
5. Click the Info icon to display the EBF/Maintenance report, or click the product description to download the software.

Conventions

The following sections describe conventions used in this manual.

SQL is a free-form language. There are no rules about the number of words you can put on a line or where you must break a line. However, for readability, all examples and most syntax statements in this manual are formatted so that each clause of a statement begins on a new line. Clauses that have more than one part extend to additional lines, which are indented. Complex commands are formatted using modified Backus Naur Form (BNF) notation.

Table 1 shows the conventions for syntax statements that appear in this manual:

<table>
<thead>
<tr>
<th>Element</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command names, procedure names, utility names, and other keywords display in sans serif font.</td>
<td><code>select</code> <code>sp_configure</code></td>
</tr>
<tr>
<td>Database names and datatypes are in sans serif font.</td>
<td><code>master database</code></td>
</tr>
<tr>
<td>Book names, file names, variables, and path names are in italics.</td>
<td><code>System Administration Guide</code></td>
</tr>
<tr>
<td></td>
<td><code>sql.ini file</code></td>
</tr>
<tr>
<td></td>
<td><code>column_name</code></td>
</tr>
<tr>
<td></td>
<td><code>$SYBASE/ASE</code> directory</td>
</tr>
</tbody>
</table>

Reference Manual: Building Blocks xix
Syntax statements (displaying the syntax and all options for a command) appear as follows:

```sql
sp_dropdevice [device_name]
```

For a command with more options:

```sql
select column_name
from table_name
where search_conditions
```

In syntax statements, keywords (commands) are in normal font and identifiers are in lowercase. Italic font shows user-supplied words.

- Syntax statements (displaying the syntax and all options for a command) appear as follows:
  ```sql
  sp_dropdevice [device_name]
  ```
  For a command with more options:
  ```sql
  select column_name
  from table_name
  where search_conditions
  ```
  In syntax statements, keywords (commands) are in normal font and identifiers are in lowercase. Italic font shows user-supplied words.

- Examples showing the use of Transact-SQL commands are printed like this:
  ```sql
  select * from publishers
  ```

- Examples of output from the computer appear as follows:
In this manual, most of the examples are in lowercase. However, you can disregard case when typing Transact-SQL keywords. For example, SELECT, Select, and select are the same.

Adaptive Server sensitivity to the case of database objects, such as table names, depends on the sort order installed on Adaptive Server. You can change case sensitivity for single-byte character sets by reconfiguring the Adaptive Server sort order. For more information, see the System Administration Guide.

This document is available in an HTML version that is specialized for accessibility. You can navigate the HTML with an adaptive technology such as a screen reader, or view it with a screen enlarger.

Adaptive Server HTML documentation has been tested for compliance with U.S. government Section 508 Accessibility requirements. Documents that comply with Section 508 generally also meet non-U.S. accessibility guidelines, such as the World Wide Web Consortium (W3C) guidelines for Web sites.

Note You might need to configure your accessibility tool for optimal use. Some screen readers pronounce text based on its case; for example, they pronounce ALL UPPERCASE TEXT as initials, and MixedCase Text as words. You might find it helpful to configure your tool to announce syntax conventions. Consult the documentation for your tool.

For information about how Sybase supports accessibility, see Sybase Accessibility at http://www.sybase.com/accessibility. The Sybase Accessibility site includes links to information on Section 508 and W3C standards.

If you need help Each Sybase installation that has purchased a support contract has one or more designated people who are authorized to contact Sybase Technical Support. If you cannot resolve a problem using the manuals or online help, please have the designated person contact Sybase Technical Support or the Sybase subsidiary in your area.
CHAPTER 1
System and User-Defined Datatypes

This chapter describes the Transact-SQL datatypes, which specify the type, size, and storage format of columns, stored procedure parameters, and local variables.

### Topics

<table>
<thead>
<tr>
<th>Topics</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Datatype categories</td>
<td>1</td>
</tr>
<tr>
<td>Range and storage size</td>
<td>2</td>
</tr>
<tr>
<td>Datatypes of columns, variables, or parameters</td>
<td>4</td>
</tr>
<tr>
<td>Datatypes of mixed-mode expressions</td>
<td>7</td>
</tr>
<tr>
<td>Datatype conversions</td>
<td>9</td>
</tr>
<tr>
<td>Standards and compliance</td>
<td>11</td>
</tr>
<tr>
<td>Exact numeric datatypes</td>
<td>12</td>
</tr>
<tr>
<td>Approximate numeric datatypes</td>
<td>16</td>
</tr>
<tr>
<td>Money datatypes</td>
<td>18</td>
</tr>
<tr>
<td>Timestamp datatype</td>
<td>19</td>
</tr>
<tr>
<td>Date and time datatypes</td>
<td>20</td>
</tr>
<tr>
<td>Character datatypes</td>
<td>27</td>
</tr>
<tr>
<td>Binary datatypes</td>
<td>32</td>
</tr>
<tr>
<td>bit datatype</td>
<td>35</td>
</tr>
<tr>
<td>sysname and longsysname datatypes</td>
<td>35</td>
</tr>
<tr>
<td>text, image, and unitext datatypes</td>
<td>36</td>
</tr>
<tr>
<td>User-defined datatypes</td>
<td>44</td>
</tr>
</tbody>
</table>

### Datatype categories

Adaptive Server provides several system datatypes and the user-defined datatypes timestamp, sysname, and longsysname. Table 1-1 lists the categories of Adaptive Server datatypes. Each category is described in a section of this chapter.
Range and storage size

Table 1-1: Datatype categories

<table>
<thead>
<tr>
<th>Category</th>
<th>Used for</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact numeric datatypes</td>
<td>Numeric values (both integers and numbers with a decimal portion) that must be represented exactly</td>
</tr>
<tr>
<td>Approximate numeric datatypes</td>
<td>Numeric data that can tolerate rounding during arithmetic operations</td>
</tr>
<tr>
<td>Money datatypes</td>
<td>Monetary data</td>
</tr>
<tr>
<td>Timestamp datatype</td>
<td>Tables that are browsed in Client-Library™ applications</td>
</tr>
<tr>
<td>Date and time datatypes</td>
<td>Date and time information</td>
</tr>
<tr>
<td>Character datatypes</td>
<td>Strings consisting of letters, numbers, and symbols</td>
</tr>
<tr>
<td>Binary datatypes</td>
<td>Raw binary data, such as pictures, in a hexadecimal-like notation</td>
</tr>
<tr>
<td>bit datatype</td>
<td>True/false and yes/no type data</td>
</tr>
<tr>
<td>sysname and longsysname</td>
<td>System tables</td>
</tr>
<tr>
<td>datatypes</td>
<td></td>
</tr>
<tr>
<td>text, image, and unitext</td>
<td>Printable characters or hexadecimal-like data that requires more than the maximum column size provided by your server's logical page size.</td>
</tr>
<tr>
<td>datatypes</td>
<td></td>
</tr>
<tr>
<td>Abstract datatypes</td>
<td>Adaptive Server supports abstract datatypes through Java classes. See Java in Adaptive Server Enterprise for more information.</td>
</tr>
<tr>
<td>User-defined datatypes</td>
<td>Defining objects that inherit the rules, default, null type, IDENTITY property, and base datatype of the datatypes listed in this table. text undergoes character-set conversion if client is using a different character set, image does not.</td>
</tr>
</tbody>
</table>

Range and storage size

Table 1-2 lists the system-supplied datatypes and their synonyms and provides information about the range of valid values and storage size for each. For simplicity, the datatypes are printed in lowercase characters, although Adaptive Server allows you to use either uppercase or lowercase characters for system datatypes. User-defined datatypes, such as timestamp, are case-sensitive. Most Adaptive Server-supplied datatypes are not reserved words and can be used to name other objects.

Table 1-2: Adaptive Server system datatypes

<table>
<thead>
<tr>
<th>Datatypes by category</th>
<th>Synonyms</th>
<th>Range</th>
<th>Bytes of storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact numeric: integers</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Datatypes by category

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Synonyms</th>
<th>Range</th>
<th>Bytes of storage</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>bigint</strong></td>
<td></td>
<td>Whole numbers between (2^{63}-1) and (-2^{63}-1) (from -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807, inclusive.)</td>
<td>8</td>
</tr>
<tr>
<td><strong>int</strong></td>
<td>integer</td>
<td>(2^{31}-1) (2,147,483,647) to (-2^{31}) (-2,147,483,648)</td>
<td>4</td>
</tr>
<tr>
<td><strong>smallint</strong></td>
<td></td>
<td>(2^{15}-1) (32,767) to (-2^{15}) (-32,768)</td>
<td>2</td>
</tr>
<tr>
<td><strong>tinyint</strong></td>
<td></td>
<td>0 to 255 (Negative numbers are not permitted)</td>
<td>1</td>
</tr>
<tr>
<td><strong>unsigned bigint</strong></td>
<td></td>
<td>Whole numbers between 0 and 18,446,744,073,709,551,615</td>
<td>8</td>
</tr>
<tr>
<td><strong>unsigned int</strong></td>
<td></td>
<td>Whole numbers between 0 and 4,294,967,295</td>
<td>4</td>
</tr>
<tr>
<td><strong>unsigned smallint</strong></td>
<td></td>
<td>Whole numbers between 0 and 65535</td>
<td>2</td>
</tr>
<tr>
<td><strong>Exact numeric: decimals</strong></td>
<td></td>
<td>(10^{38}) -1 to (-10^{38})</td>
<td>2 to 17</td>
</tr>
<tr>
<td><strong>decimal (p, s)</strong></td>
<td>dec</td>
<td>(10^{38}) -1 to (-10^{38})</td>
<td>2 to 17</td>
</tr>
<tr>
<td><strong>Approximate numeric</strong></td>
<td></td>
<td>machine dependent</td>
<td>4 for default precision &lt; 16, 8 for default precision &gt;= 16</td>
</tr>
<tr>
<td><strong>float (precision)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>double precision</strong></td>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td><strong>real</strong></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td><strong>Money</strong></td>
<td></td>
<td>(214,748.3647) to (-214,748.3648)</td>
<td>4</td>
</tr>
<tr>
<td><strong>smallmoney</strong></td>
<td></td>
<td>(922,337,203.685,477,5807) to (-922,337,203.685,477,5808)</td>
<td>8</td>
</tr>
<tr>
<td><strong>money</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Date/Time</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>smalldatetime</strong></td>
<td></td>
<td>January 1, 1900 to June 6, 2079</td>
<td>4</td>
</tr>
<tr>
<td><strong>datetime</strong></td>
<td></td>
<td>January 1, 1753 to December 31, 9999</td>
<td>8</td>
</tr>
<tr>
<td><strong>date</strong></td>
<td></td>
<td>January 1, 0001 to December 31, 9999</td>
<td>4</td>
</tr>
<tr>
<td><strong>time</strong></td>
<td></td>
<td>12:00:00AM to 11:59:59:999PM</td>
<td>4</td>
</tr>
<tr>
<td><strong>bigdatetime</strong></td>
<td></td>
<td>January 1, 0001 to December 31, 9999 and 12:00:000000AM to 11:59:59,999999 PM</td>
<td>8</td>
</tr>
</tbody>
</table>
You must declare the datatype for a column, local variable, or parameter. The datatype can be any of the system-supplied datatypes, or any user-defined datatype in the database.

### Datatypes of columns, variables, or parameters

<table>
<thead>
<tr>
<th>Datatypes by category</th>
<th>Synonyms</th>
<th>Range</th>
<th>Bytes of storage</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Character</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bigtime</td>
<td></td>
<td>12:00:00.000000 AM to 11:59:59.999999 PM</td>
<td>8</td>
</tr>
<tr>
<td>Character</td>
<td>character</td>
<td>pagesize</td>
<td>n</td>
</tr>
<tr>
<td>varchar(n)</td>
<td>character varying, char varying</td>
<td>pagesize</td>
<td>actual entry length</td>
</tr>
<tr>
<td>unichar</td>
<td>Unicode character</td>
<td>pagesize</td>
<td>n * @@unicharsize ( @@unicharsize equals 2)</td>
</tr>
<tr>
<td>univarchar</td>
<td>Unicode character varying, char varying</td>
<td>pagesize</td>
<td>actual number of characters * @@unicharsize</td>
</tr>
<tr>
<td>nchar(n)</td>
<td>national character, national char</td>
<td>pagesize</td>
<td>n * @@ncharsize</td>
</tr>
<tr>
<td>nvchar(n)</td>
<td>nchar varying, national char varying</td>
<td>pagesize</td>
<td>@@ncharsize * number of characters</td>
</tr>
<tr>
<td>text</td>
<td></td>
<td>2^31 -1 (2,147,483,647) bytes or fewer</td>
<td>0 when uninitialized; multiple of 2K after initialization</td>
</tr>
<tr>
<td>untext</td>
<td></td>
<td>1 – 1,073,741,823</td>
<td>0 when uninitialized; multiple of 2K after initialization</td>
</tr>
<tr>
<td><strong>Binary</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>binary(n)</td>
<td></td>
<td>pagesize</td>
<td>n</td>
</tr>
<tr>
<td>varbinary(n)</td>
<td></td>
<td>pagesize</td>
<td>actual entry length</td>
</tr>
<tr>
<td>image</td>
<td></td>
<td>2^31 -1 (2,147,483,647) bytes or fewer</td>
<td>0 when uninitialized; multiple of 2K after initialization</td>
</tr>
<tr>
<td><strong>Bit</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bit</td>
<td></td>
<td>0 or 1</td>
<td>1 (one byte holds up to 8 bit columns)</td>
</tr>
</tbody>
</table>
Declaring the datatype for a column in a table

To declare the datatype of a new column in a create table or alter table statement, use:

```
create table [[database.]owner.]table_name
    (column_name datatype [identity | not null | null]...
    [, column_name datatype [identity | not null | null]]...)

alter table [[database.]owner.]table_name
    add column_name datatype [identity | null...
    [, column_name datatype [identity | null]...]
```

For example:

```
create table sales_daily
    (stor_id char(4) not null, 
    ord_num numeric(10,0) identity, 
    ord_amt money null)
```

You can also declare the datatype of a new column in a select into statement, use `convert` or `cast`:

```
select convert(double precision, x), cast (int, y) into newtable from oldtable
```

Declaring the datatype for a local variable in a batch or procedure

To declare the datatype for a local variable in a batch or stored procedure, use:

```
declare @variable_name datatype 
    [, @variable_name datatype ]...
```

For example:

```
declare @hope money
```

Declaring the datatype for a parameter in a stored procedure

Use the following syntax to declare the datatype for a parameter in a stored procedure:

```
create procedure [owner.]procedure_name [;n
umber]
    ([@]parameter_name datatype [= default] [output] 
    [, @parameter_name datatype [= default] [output]...])

[with recompile]
as SQL_statements
```
Datatypes of columns, variables, or parameters

For example:

```sql
create procedure auname_sp @auname varchar(40) 
as
    select au_lname, title, au_ord 
    from authors, titles, titleauthor 
    where @auname = au_lname 
    and authors.au_id = titleauthor.au_id 
    and titles.title_id = titleauthor.title_id
```

Determining the datatype of a literal

Numeric literals

Numeric literals entered with E notation are treated as float; all others are treated as exact numerics:

- Literals between $2^{31} - 1$ and $-2^{31}$ with no decimal point are treated as integer.
- Literals that include a decimal point, or that fall outside the range for integers, are treated as numeric.

**Note** To preserve backward compatibility, use E notation for numeric literals that should be treated as float.

Character literals

In versions of Adaptive Server earlier than 12.5.1, when the client’s character set was different from the server’s character set, conversions were generally enabled to allow the text of SQL queries to be converted to the server’s character set before being processed. If any character could not be converted because it could not be represented in the server’s character set, the entire query was rejected. This character set “bottleneck” has been removed as of Adaptive Server version 12.5.1.
You cannot declare the datatype of a character literal. Adaptive Server treats character literals as `varchar`, except those that contain characters that cannot be converted to the server’s default character set. Such literals are treated as `univarchar`. This makes it possible to perform such queries as selecting `unichar` data in a server configured for “iso_1” using a “sjis” (Japanese) client. For example:

```sql
SELECT * FROM mytable WHERE unichar_column = 'Hi'
```

Since the character literal cannot be represented using the `char` datatype (in “iso_1”), it is promoted to the `unichar` datatype, and the query succeeds.

### Datatypes of mixed-mode expressions

When you perform concatenation or mixed-mode arithmetic on values with different datatypes, Adaptive Server must determine the datatype, length, and precision of the result.

### Determining the datatype hierarchy

Each system datatype has a **datatype hierarchy**, which is stored in the `systypes` system table. User-defined datatypes inherit the hierarchy of the system datatype on which they are based.

The following query ranks the datatypes in a database by hierarchy. In addition to the information shown below, your query results will include information about any user-defined datatypes in the database:

```sql
SELECT name, hierarchy FROM systypes ORDER BY hierarchy
```

<table>
<thead>
<tr>
<th>name</th>
<th>hierarchy</th>
</tr>
</thead>
<tbody>
<tr>
<td>floatn</td>
<td>1</td>
</tr>
<tr>
<td>float</td>
<td>2</td>
</tr>
<tr>
<td>datetimn</td>
<td>3</td>
</tr>
<tr>
<td>datetime</td>
<td>4</td>
</tr>
<tr>
<td>real</td>
<td>5</td>
</tr>
<tr>
<td>numericn</td>
<td>6</td>
</tr>
<tr>
<td>numeric</td>
<td>7</td>
</tr>
</tbody>
</table>

Reference Manual: Building Blocks
Datatypes of mixed-mode expressions

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>decimaln</td>
<td>8</td>
</tr>
<tr>
<td>decimal</td>
<td>9</td>
</tr>
<tr>
<td>moneyyn</td>
<td>10</td>
</tr>
<tr>
<td>money</td>
<td>11</td>
</tr>
<tr>
<td>smallmoney</td>
<td>12</td>
</tr>
<tr>
<td>smalldatet</td>
<td>13</td>
</tr>
<tr>
<td>intn</td>
<td>14</td>
</tr>
<tr>
<td>uintn</td>
<td>15</td>
</tr>
<tr>
<td>bigint</td>
<td>16</td>
</tr>
<tr>
<td>ubigint</td>
<td>17</td>
</tr>
<tr>
<td>int</td>
<td>18</td>
</tr>
<tr>
<td>uint</td>
<td>19</td>
</tr>
<tr>
<td>smallint</td>
<td>20</td>
</tr>
<tr>
<td>usmallint</td>
<td>21</td>
</tr>
<tr>
<td>tinyint</td>
<td>22</td>
</tr>
<tr>
<td>bit</td>
<td>23</td>
</tr>
<tr>
<td>univarchar</td>
<td>24</td>
</tr>
<tr>
<td>unichar</td>
<td>25</td>
</tr>
<tr>
<td>unitext</td>
<td>26</td>
</tr>
<tr>
<td>sysname</td>
<td>27</td>
</tr>
<tr>
<td>varchar</td>
<td>27</td>
</tr>
<tr>
<td>nvarchar</td>
<td>27</td>
</tr>
<tr>
<td>longsysnam</td>
<td>27</td>
</tr>
<tr>
<td>char</td>
<td>28</td>
</tr>
<tr>
<td>nchar</td>
<td>28</td>
</tr>
<tr>
<td>timestamp</td>
<td>29</td>
</tr>
<tr>
<td>varbinary</td>
<td>29</td>
</tr>
<tr>
<td>binary</td>
<td>30</td>
</tr>
<tr>
<td>text</td>
<td>31</td>
</tr>
<tr>
<td>image</td>
<td>32</td>
</tr>
<tr>
<td>date</td>
<td>33</td>
</tr>
<tr>
<td>time</td>
<td>34</td>
</tr>
<tr>
<td>daten</td>
<td>35</td>
</tr>
<tr>
<td>timen</td>
<td>36</td>
</tr>
<tr>
<td>bigdatetim</td>
<td>37</td>
</tr>
<tr>
<td>bigtime</td>
<td>38</td>
</tr>
<tr>
<td>bigdatetim</td>
<td>39</td>
</tr>
<tr>
<td>bigtimen</td>
<td>40</td>
</tr>
<tr>
<td>extended t</td>
<td>99</td>
</tr>
</tbody>
</table>

**Note**  
*u-int type* is an internal representation. The correct syntax for unsigned types is unsigned {int | integer | bigint | smallint}
The datatype hierarchy determines the results of computations using values of different datatypes. The result value is assigned the datatype that is closest to the top of the list or has the least hierarchical value.

In the following example, \( qty \) from the sales table is multiplied by royalty from the roysched table. \( qty \) is a smallint, which has a hierarchy of 20; royalty is an int, which has a hierarchy of 18. Therefore, the datatype of the result is an int:

\[
\text{smallint}(qty) \times \text{int}(royalty) = \text{int}
\]

**Determining precision and scale**

For numeric and decimal datatypes, each combination of precision and scale is a distinct Adaptive Server datatype. If you perform arithmetic on two numeric or decimal values:

- \( n1 \) with precision \( p1 \) and scale \( s1 \), and
- \( n2 \) with precision \( p2 \) and scale \( n2 \)

Adaptive Server determines the precision and scale of the results as shown in Table 1-3.

**Table 1-3: Precision and scale after arithmetic operations**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Precision</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n1 + n2 )</td>
<td>( \max(s1, s2) + \max(p1 -s1, p2 - s2) + 1 )</td>
<td>( \max(s1, s2) )</td>
</tr>
<tr>
<td>( n1 - n2 )</td>
<td>( \max(s1, s2) + \max(p1 -s1, p2 - s2) + 1 )</td>
<td>( \max(s1, s2) )</td>
</tr>
<tr>
<td>( n1 \times n2 )</td>
<td>( s1 + s2 + (p1 - s1) + (p2 - s2) + 1 )</td>
<td>( s1 + s2 )</td>
</tr>
<tr>
<td>( n1 / n2 )</td>
<td>( \max(s1 + p2 + 1, 6) + p1 - s1 + p2 )</td>
<td>( \max(s1 + p2 -s2 + 1, 6) )</td>
</tr>
</tbody>
</table>

**Datatype conversions**

Many conversions from one datatype to another are handled automatically by Adaptive Server. These are called implicit conversions. Other conversions must be performed explicitly with the `convert`, `hextoint`, `intohex`, `hextobigint`, and `bigintohex` functions. See “Datatype conversion functions” on page 63 for details about datatype conversions supported by Adaptive Server.
Datatype conversions

**Automatic conversion of fixed-length NULL columns**

Only columns with variable-length datatypes can store null values. When you create a NULL column with a fixed-length datatype, Adaptive Server automatically converts it to the corresponding variable-length datatype. Adaptive Server does not inform the user of the datatype change.

Table 1-4 lists the fixed- and variable-length datatypes to which they are converted. Certain variable-length datatypes, such as moneyn, are reserved datatypes; you cannot use them to create columns, variables, or parameters:

<table>
<thead>
<tr>
<th>Original fixed-length datatype</th>
<th>Converted to</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>varchar</td>
</tr>
<tr>
<td>unichar</td>
<td>univarchar</td>
</tr>
<tr>
<td>nchar</td>
<td>nvarchar</td>
</tr>
<tr>
<td>binary</td>
<td>varbinary</td>
</tr>
<tr>
<td>datetime</td>
<td>datetimn</td>
</tr>
<tr>
<td>date</td>
<td>daten</td>
</tr>
<tr>
<td>time</td>
<td>timen</td>
</tr>
<tr>
<td>float</td>
<td>floatn</td>
</tr>
<tr>
<td>bigint, int, smallint, and tinyint</td>
<td>intn</td>
</tr>
<tr>
<td>unsigned bigint, unsigned int, and unsigned smallint</td>
<td>uintn</td>
</tr>
<tr>
<td>decimal</td>
<td>decimaln</td>
</tr>
<tr>
<td>numeric</td>
<td>numericn</td>
</tr>
<tr>
<td>money and smallmoney</td>
<td>moneyn</td>
</tr>
</tbody>
</table>

**Handling overflow and truncation errors**

The arithabort option determines how Adaptive Server behaves when an arithmetic error occurs. The two arithabort options, arithabort arith_overflow and arithabort numeric_truncation, handle different types of arithmetic errors. You can set each option independently, or set both options with a single set arithabort on or set arithabort off statement.
• arithabort arith_overflow specifies behavior following a divide-by-zero error or a loss of precision during either an explicit or an implicit datatype conversion. This type of error is considered serious. The default setting, arithabort arith_overflow on, rolls back the entire transaction in which the error occurs. If the error occurs in a batch that does not contain a transaction, arithabort arith_overflow on does not roll back earlier commands in the batch, but Adaptive Server does not execute any statements that follow the error-generating statement in the batch.

Setting arith_overflow to on refers to the execution time, not to the level of normalization to which Adaptive Server is set.

If you set arithabort arith_overflow off, Adaptive Server aborts the statement that causes the error, but continues to process other statements in the transaction or batch.

• arithabort numeric_truncation specifies behavior following a loss of scale by an exact numeric datatype during an implicit datatype conversion. (When an explicit conversion results in a loss of scale, the results are truncated without warning.) The default setting, arithabort numeric_truncation on, aborts the statement that causes the error but continues to process other statements in the transaction or batch. If you set arithabort numeric_truncation off, Adaptive Server truncates the query results and continues processing.

The arithignore option determines whether Adaptive Server prints a warning message after an overflow error. By default, the arithignore option is turned off. This causes Adaptive Server to display a warning message after any query that results in numeric overflow. To ignore overflow errors, use set arithignore on.

Standards and compliance

Table 1-5 lists the ANSI SQL standards and compliance levels for Transact-SQL datatypes.
Exact numeric datatypes

Use the exact numeric datatypes when you must represent a value exactly. Adaptive Server provides exact numeric types for both integers (whole numbers) and numbers with a decimal portion.

<table>
<thead>
<tr>
<th>Transact-SQL – ANSI SQL datatypes</th>
<th>Transact-SQL extensions – User-defined datatypes</th>
</tr>
</thead>
<tbody>
<tr>
<td>• char</td>
<td>• binary</td>
</tr>
<tr>
<td>• varchar</td>
<td>• varbinary</td>
</tr>
<tr>
<td>• smallint</td>
<td>• bit</td>
</tr>
<tr>
<td>• int</td>
<td>• nchar</td>
</tr>
<tr>
<td>• bigint</td>
<td>• datetime</td>
</tr>
<tr>
<td>• decimal</td>
<td>• smalldatetime</td>
</tr>
<tr>
<td>• numeric</td>
<td>• bigdatetime</td>
</tr>
<tr>
<td>• float</td>
<td>• bigtime</td>
</tr>
<tr>
<td>• real</td>
<td>• tinyint</td>
</tr>
<tr>
<td>• date</td>
<td>• unsigned smallint</td>
</tr>
<tr>
<td>• time</td>
<td>• unsigned int</td>
</tr>
<tr>
<td>• double precision</td>
<td>• unsigned bigint</td>
</tr>
<tr>
<td></td>
<td>• money</td>
</tr>
<tr>
<td></td>
<td>• smallmoney</td>
</tr>
<tr>
<td></td>
<td>• text</td>
</tr>
<tr>
<td></td>
<td>• unitext</td>
</tr>
<tr>
<td></td>
<td>• image</td>
</tr>
<tr>
<td></td>
<td>• nvarchar</td>
</tr>
<tr>
<td></td>
<td>• unichar</td>
</tr>
<tr>
<td></td>
<td>• univarchar</td>
</tr>
<tr>
<td></td>
<td>• sysname</td>
</tr>
<tr>
<td></td>
<td>• longsysname</td>
</tr>
<tr>
<td></td>
<td>• timestamp</td>
</tr>
</tbody>
</table>
CHAPTER 1  System and User-Defined Datatypes

Integer types

Adaptive Server provides the following exact numeric datatypes to store integers: bigint, int (or integer), smallint, tinyint and each of their unsigned counterparts. Choose the integer type based on the expected size of the numbers to be stored. Internal storage size varies by type, as shown in Table 1-6.

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Stores</th>
<th>Bytes of storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>bigint</td>
<td>Whole numbers between -2^{63} and 2^{63} - 1 (from -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807), inclusive.</td>
<td>8</td>
</tr>
<tr>
<td>int[eger]</td>
<td>Whole numbers between -2^{31} and 2^{31} - 1 (-2,147,483,648 and 2,147,483,647), inclusive.</td>
<td>4</td>
</tr>
<tr>
<td>smallint</td>
<td>Whole numbers between -2^{15} and 2^{15} - 1 (-32,768 and 32,767), inclusive.</td>
<td>2</td>
</tr>
<tr>
<td>tinyint</td>
<td>Whole numbers between 0 and 255, inclusive. (Negative numbers are not permitted.)</td>
<td>1</td>
</tr>
<tr>
<td>unsigned bigint</td>
<td>Whole numbers between 0 and 18,446,744,073,709,551,615</td>
<td>8</td>
</tr>
<tr>
<td>unsigned int</td>
<td>Whole numbers between 0 and 4,294,967,295</td>
<td>4</td>
</tr>
<tr>
<td>unsigned smallint</td>
<td>Whole numbers between 0 and 65,535</td>
<td>2</td>
</tr>
</tbody>
</table>

Entering integer data

Enter integer data as a string of digits without commas. Integer data can include a decimal point as long as all digits to the right of the decimal point are zeros. The smallint, integer, and bigint datatypes can be preceded by an optional plus or minus sign. The tinyint datatype can be preceded by an optional plus sign.

Table 1-7 shows some valid entries for a column with a datatype of integer and indicates how isql displays these values:

<table>
<thead>
<tr>
<th>Value entered</th>
<th>Value displayed</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>+2</td>
<td>2</td>
</tr>
<tr>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>2.</td>
<td>2</td>
</tr>
<tr>
<td>2.000</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 1-8 lists some invalid entries for an integer column:
**Exact numeric datatypes**

<table>
<thead>
<tr>
<th>Value entered</th>
<th>Type of error</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,000</td>
<td>Commas not allowed.</td>
</tr>
<tr>
<td>2-</td>
<td>Minus sign should precede digits.</td>
</tr>
<tr>
<td>3.45</td>
<td>Digits to the right of the decimal point are nonzero digits.</td>
</tr>
</tbody>
</table>

### Decimal datatypes

Adaptive Server provides two other exact numeric datatypes, numeric and dec[imal], for numbers that include decimal points. The numeric and decimal datatypes are identical in all respects but one: only numeric datatypes with a scale of 0 and integer datatypes can be used for the IDENTITY column.

#### Specifying precision and scale

The numeric and decimal datatypes accept two optional parameters, precision and scale, enclosed in parentheses and separated by a comma:

```
datatype [(precision [, scale])]
```

Adaptive Server treats each combination of precision and scale as a distinct datatype. For example, numeric(10,0) and numeric(5,0) are two separate datatypes. The precision and scale determine the range of values that can be stored in a decimal or numeric column:

- The precision specifies the maximum number of decimal digits that can be stored in the column. It includes all digits, both to the right and to the left of the decimal point. You can specify precisions ranging from 1 digit to 38 digits or use the default precision of 18 digits.
- The scale specifies the maximum number of digits that can be stored to the right of the decimal point. The scale must be less than or equal to the precision. You can specify a scale ranging from 0 digits to 38 digits, or use the default scale of 0 digits.

#### Storage size

The storage size for a numeric or decimal column depends on its precision. The minimum storage requirement is 2 bytes for a 1- or 2-digit column. Storage size increases by approximately 1 byte for each additional 2 digits of precision, up to a maximum of 17 bytes.

Use the following formula to calculate the exact storage size for a numeric or decimal column:

```
ceiling (precision / log10(256)) + 1
```

For example, the storage size for a numeric(18,4) column is 9 bytes.

14 Adaptive Server Enterprise
Entering decimal data

Enter decimal and numeric data as a string of digits preceded by an optional plus or minus sign and including an optional decimal point. If the value exceeds either the precision or scale specified for the column, Adaptive Server returns an error message. Exact numeric types with a scale of 0 are displayed without a decimal point.

Table 1-9 shows some valid entries for a column with a datatype of numeric(5,3) and indicates how these values are displayed by isql:

<table>
<thead>
<tr>
<th>Value entered</th>
<th>Value displayed</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.345</td>
<td>12.345</td>
</tr>
<tr>
<td>+12.345</td>
<td>12.345</td>
</tr>
<tr>
<td>-12.345</td>
<td>-12.345</td>
</tr>
<tr>
<td>12.345000</td>
<td>12.345</td>
</tr>
<tr>
<td>12.1</td>
<td>12.100</td>
</tr>
<tr>
<td>12</td>
<td>12.000</td>
</tr>
</tbody>
</table>

Table 1-10 shows some invalid entries for a column with a datatype of numeric(5,3):

<table>
<thead>
<tr>
<th>Value entered</th>
<th>Type of error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,200</td>
<td>Commas not allowed.</td>
</tr>
<tr>
<td>12-</td>
<td>Minus sign should precede digits.</td>
</tr>
<tr>
<td>12.345678</td>
<td>Too many nonzero digits to the right of the decimal point.</td>
</tr>
</tbody>
</table>

Standards and compliance

Transact-SQL provides the smallint, int, bigint, numeric, and decimal ANSI SQL exact numeric datatypes. The unsigned bigint, unsigned int, unsigned smallint, and tinyint type is a Transact-SQL extension.
Approximate numeric datatypes

Use the approximate numeric types, float, double precision, and real, for numeric data that can tolerate rounding. The approximate numeric types are especially suited to data that covers a wide range of values. They support all aggregate functions and all arithmetic operations.

Understanding approximate numeric datatypes

Approximate numeric datatypes, used to store floating-point numbers, are inherently slightly inaccurate in their representation of real numbers—hence the name “approximate numeric.” To use these datatypes, you must understand their limitations.

When a floating-point number is printed or displayed, the printed representation is not quite the same as the stored number, and the stored number is not quite the same as the number that the user entered. Most of the time, the stored representation is close enough, and software makes the printed output look just like the original input, but you must understand the inaccuracy if you plan to use floating-point numbers for calculations, particularly if you are doing repeated calculations using approximate numeric datatypes—the results can be surprisingly and unexpectedly inaccurate.

The inaccuracy occurs because floating-point numbers are stored in the computer as binary fractions (that is, as a representative number divided by a power of 2), but the numbers we use are decimal (powers of 10). This means that only a very small set of numbers can be stored accurately: 0.75 (3/4) can be stored accurately because it is a binary fraction (4 is a power of 2); 0.2 (2/10) cannot (10 is not a power of 2).

Some numbers contain too many digits to store accurately. double precision is stored as 8 binary bytes and can represent about 17 digits with reasonable accuracy. real is stored as 4 binary bytes and can represent only about 6 digits with reasonable accuracy.

If you begin with numbers that are almost correct, and perform computations with them using other numbers that are almost correct, you can easily end up with a result that is not even close to being correct. If these considerations are important to your application, use an exact numeric datatype.
Range, precision, and storage size

The real and double precision types are built on types supplied by the operating system. The float type accepts an optional binary precision in parentheses. float columns with a precision of 1–15 are stored as real; those with higher precision are stored as double precision.

The range and storage precision for all three types is machine-dependent.

Table 1-11 shows the range and storage size for each approximate numeric type. isql displays only 6 significant digits after the decimal point and rounds the remainder:

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Bytes of storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>float[default precision]</td>
<td>4 for default precision ≤ 16</td>
</tr>
<tr>
<td></td>
<td>8 for default precision ≥ 16</td>
</tr>
<tr>
<td>double precision</td>
<td>8</td>
</tr>
<tr>
<td>real</td>
<td>4</td>
</tr>
</tbody>
</table>

Entering approximate numeric data

Enter approximate numeric data as a mantissa followed by an optional exponent:

- The mantissa is a signed or unsigned number, with or without a decimal point. The column’s binary precision determines the maximum number of binary digits allowed in the mantissa.
- The exponent, which begins with the character “e” or “E,” must be a whole number.

The value represented by the entry is the following product:

\[ \text{mantissa} \times 10^{\text{EXPONENT}} \]

For example, 2.4E3 represents the value 2.4 times 10^3, or 2400.
**NaN and Inf values**

“NaN” and “Inf” are special values that the IEEE754/854 floating point number standards use to represent values that are “not a number” and “infinity,” respectively. In accordance with the ANSI SQL92 standard, Adaptive Server versions 12.5 and later do not allow the insertion of these values in the database and do not allow them to be generated. In Adaptive Server versions earlier than 12.5, Open Client clients such as native-mode bcp, JDBC, and ODBC could occasionally force these values into tables.

If you encounter a NaN or an Inf value in the database, contact Sybase Customer Support with details of how to reproduce the problem.

**Standards and compliance**

ANSI SQL – Compliance level: The float, double precision, and real datatypes are entry-level compliant.

**Money datatypes**

Use the money and smallmoney datatypes to store monetary data. You can use these types for U.S. dollars and other decimal currencies, but Adaptive Server provides no means to convert from one currency to another. You can use all arithmetic operations except modulo, and all aggregate functions, with money and smallmoney data.

**Accuracy**

Both money and smallmoney are accurate to one ten-thousandth of a monetary unit, but they round values up to two decimal places for display purposes. The default print format places a comma after every three digits.

**Range and storage size**

Table 1-12 summarizes the range and storage requirements for money datatypes:
Table 1-12: Money datatypes

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Range</th>
<th>Bytes of storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>money</td>
<td>Monetary values between +922,337,203,685,477,5807 and -922,337,203,685,477,5808</td>
<td>8</td>
</tr>
<tr>
<td>smallmoney</td>
<td>Monetary values between +214,748.3647 and -214,748.3648</td>
<td>4</td>
</tr>
</tbody>
</table>

### Entering monetary values
Monetary values entered with E notation are interpreted as float. This may cause an entry to be rejected or to lose some of its precision when it is stored as a money or smallmoney value.

Money and smallmoney values can be entered with or without a preceding currency symbol, such as the dollar sign ($), yen sign (¥), or pound sterling sign (£). To enter a negative value, place the minus sign after the currency symbol. Do not include commas in your entry.

### Standards and compliance
ANSI SQL – The money and smallmoney datatypes are Transact-SQL extensions.

### Timestamp datatype
Use the user-defined timestamp datatype in tables that are to be browsed in Client-Library™ applications (see “Browse Mode” for more information). Adaptive Server updates the timestamp column each time its row is modified. A table can have only one column of timestamp datatype.

### Creating a timestamp column
If you create a column named timestamp without specifying a datatype, Adaptive Server defines the column as a timestamp datatype:

```sql
create table testing
    (c1 int, timestamp, c2 int)
```
Date and time datatypes

You can also explicitly assign the timestamp datatype to a column named timestamp:

```sql
create table testing
  (c1 int, timestamp timestamp, c2 int)
```
or to a column with another name:

```sql
create table testing
  (c1 int, t_stamp timestamp, c2 int)
```
You can create a column named timestamp and assign it another datatype (although this may be confusing to other users and does not allow the use of the `browse` functions in Open Client™ or with the `tsequal` function):

```sql
create table testing
  (c1 int, timestamp datetime)
```

Date and time datatypes

Use `datetime`, `smalldatetime`, `bigdatetime`, `bigtime`, `date`, and `time` to store absolute date and time information. Use `timestamp` to store binary-type information.

Adaptive Server has various datatypes used to store date and time values.

- **date**
- **time**
- **smalldatetime**
- **datetime**
- **bigdatetime**
- **bigtime**

The default display format for dates is “Apr 15 1987 10:23PM”. `bigdatetime/bigtime` types have a default display format of “Apr 15 1987 10:23:00.000000PM”. You can use the `convert` function for other styles of date display. You can also perform some arithmetic calculations on date and time values with the built-in date functions, though Adaptive Server may round or truncate millisecond values.
datetime columns hold dates between January 1, 1753 and December 31, 9999. datetime values are accurate to 1/300 second on platforms that support this level of granularity. Storage size is 8 bytes: 4 bytes for the number of days since the base date of January 1, 1900 and 4 bytes for the time of day.

smalldatetime columns hold dates from January 1, 1900 to June 6, 2079, with accuracy to the minute. Its storage size is 4 bytes: 2 bytes for the number of days after January 1, 1900, and 2 bytes for the number of minutes after midnight.

bigdatetime columns hold dates from January 1, 0001 to December 31, 9999 and 12:00:00.000000 AM to 11:59:59.999999 PM. Its storage size is 8 bytes. The internal representation of bigdatetime is a 64 bit integer containing the number of microseconds since 01/01/0000.

bigtime columns hold times from 12:00:00.000000 AM to 11:59:59.999999 PM. Its storage size is 8 bytes. The internal representation of bigtime is a 64 bit integer containing the number of microseconds since midnight.

date columns hold dates from January 1, 0001 to December 31, 9999. Storage size is 4 bytes.

time is between 00:00:00:000 and 23:59:59:999. You can use either military time or 12AM for noon and 12PM for midnight. A time value must contain either a colon or the AM or PM signifier. AM or PM may be in either uppercase or lowercase.

When entering date and time information, always enclose the time or date in single or double quotes.

Range and storage requirements

Table 1-13 summarizes the range and storage requirements for the datetime, smalldatetime, bigdatetime, bigtime, date, and time datatypes:
Date and time datatypes

Table 1-13: Transact-SQL datatypes for storing dates and times

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Range</th>
<th>Bytes of storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>datetime</td>
<td>January 1, 1753 through December 31, 9999</td>
<td>8</td>
</tr>
<tr>
<td>smalldatetime</td>
<td>January 1, 1900 through June 6, 2079</td>
<td>4</td>
</tr>
<tr>
<td>bigdatetime</td>
<td>January 1, 0001 to December 31, 9999</td>
<td>8</td>
</tr>
<tr>
<td>bigtime</td>
<td>12:00:00.000000AM to 11:59:59.999999PM</td>
<td>8</td>
</tr>
<tr>
<td>date</td>
<td>January 1, 0001 to December 31, 9999</td>
<td>4</td>
</tr>
<tr>
<td>time</td>
<td>12:00:00 AM to 11:59:59:999 PM</td>
<td>4</td>
</tr>
</tbody>
</table>

Entering date and time data

The datetime, smalldatetime, bigdatetime and bigtime datatypes consist of a date portion either followed by or preceded by a time portion. (You can omit either the date or the time, or both.) The date datatype has only a date and the time datatype has only the time. You must enclose values in single or double quotes.

Entering the date

Dates consist of a month, day, and year and can be entered in a variety of formats for date, datetime, bigdatetime, bigtime and smalldatetime:

- You can enter the entire date as an unseparated string of 4, 6, or 8 digits, or use slash (/), hyphen (-), or period (.) separators between the date parts.
- When entering dates as unseparated strings, use the appropriate format for that string length. Use leading zeros for single-digit years, months, and days. Dates entered in the wrong format may be misinterpreted or result in errors.
- When entering dates with separators, use the set dateformat option to determine the expected order of date parts. If the first date part in a separated string is four digits, Adaptive Server interprets the string as yyyy-mm-dd format.
- Some date formats accept 2-digit years (yy):
  - Numbers less than 50 are interpreted as 20yy. For example, 01 is 2001, 32 is 2032, and 49 is 2049.
  - Numbers equal to or greater than 50 are interpreted as 19yy. For example, 50 is 1950, 74 is 1974, and 99 is 1999.
- You can specify the month as either a number or a name. Month names and their abbreviations are language-specific and can be entered in uppercase, lowercase, or mixed case.
If you omit the date portion of a datetime or smalldatetime value, Adaptive Server uses the default date of January 1, 1900. If you omit the date portion of a bigdatetime a default value of January 1, 0001 will be added.

Table 1-14 describes the acceptable formats for entering the date portion of a datetime or smalldatetime value:

<table>
<thead>
<tr>
<th>Date format</th>
<th>Interpretation</th>
<th>Sample entries</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-digit string with no separators</td>
<td>Interpreted as yyyy. Date defaults to Jan 1 of the specified year.</td>
<td>“1947”</td>
<td>Jan 1 1947</td>
</tr>
<tr>
<td>6-digit string with no separators</td>
<td>Interpreted as yymmd. For yy &lt; 50, year is 20yy. For yy &gt;= 50, year is 19yy.</td>
<td>“450128”</td>
<td>Jan 28 2045</td>
</tr>
<tr>
<td>8-digit string with no separators</td>
<td>Interpreted as yyyyymmd.</td>
<td>“19940415”</td>
<td>Apr 15 1994</td>
</tr>
<tr>
<td>String consisting of 2-digit month, day, and year separated by slashes, hyphens, or periods, or a combination of the above</td>
<td>The dateformat and language set options determine the expected order of date parts. For us_english, the default order is mdy. For yy &lt; 50, year is interpreted as 20yy. For yy &gt;= 50, year is interpreted as 19yy.</td>
<td>“4/15/94”</td>
<td>All of these entries are interpreted as Apr 15 1994 when the dateformat option is set to mdy.</td>
</tr>
<tr>
<td>String consisting of 2-digit month, 2-digit day, and 4-digit year separated by slashes, hyphens, or periods, or a combination of the above</td>
<td>The dateformat and language set options determine the expected order of date parts. For us_english, the default order is mdy.</td>
<td>“04/15.1994”</td>
<td>Interpreted as Apr 15 1994 when the dateformat option is set to mdy.</td>
</tr>
<tr>
<td>Month is entered in character form (either full month name or its standard abbreviation), followed by an optional comma</td>
<td>If 4-digit year is entered, date parts can be entered in any order.</td>
<td>“April 15, 1994”</td>
<td>All of these entries are interpreted as Apr 15 1994.</td>
</tr>
<tr>
<td></td>
<td>If day is omitted, all 4 digits of year must be specified. Day defaults to the first day of the month.</td>
<td>“1994 15 apr”</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If year is only 2 digits (yy), it is expected to appear after the day. For yy &lt; 50, year is interpreted as 20yy. For yy &gt;= 50, year is interpreted as 19yy.</td>
<td>“1994 April 15”</td>
<td></td>
</tr>
<tr>
<td></td>
<td>“15 APR 1994”</td>
<td>“15 APR 1994”</td>
<td></td>
</tr>
<tr>
<td></td>
<td>“mar 16 17”</td>
<td>“mar 16 17”</td>
<td>Mar 16 2017</td>
</tr>
<tr>
<td></td>
<td>“apr 15 94”</td>
<td>“apr 15 94”</td>
<td>Apr 15 1994</td>
</tr>
<tr>
<td>The empty string “”</td>
<td>Date defaults to Jan 1 1900.</td>
<td>“”</td>
<td>Jan 1 1900</td>
</tr>
</tbody>
</table>

Entering the time

The time component of a datetime, smalldatetime, or time value must be specified as follows:
**Date and time datatypes**

The time component of a bigdatetime or bigtime value must be specified as follows:

```
hours[:minutes[:seconds[:milliseconds]]] [AM | PM]
```

- Use 12AM for midnight and 12PM for noon.
- A time value must contain either a colon or an AM or PM signifier. The AM or PM can be entered in uppercase, lowercase, or mixed case.
- The seconds specification can include either a decimal portion preceded by a decimal point, or a number of milliseconds preceded by a colon. For example, “15:30:20:1” means twenty seconds and one millisecond past 3:30 PM; “15:30:20.1” means twenty and one-tenth of a second past 3:30 PM. Microseconds must be expressed with a decimal point.
- If you omit the time portion of a datetime or smalldatetime value, Adaptive Server uses the default time of 12:00:00:000AM.

The display format for datetime and smalldatetime values is “Mon dd yyyy hh:mmAM” (or “PM”); for example, “Apr 15 1988 10:23PM”. To display seconds and milliseconds, and to obtain additional date styles and date-part orders, use the convert function to convert the data to a character string. Adaptive Server may round or truncate millisecond values.

Table 1-15 lists some examples of datetime entries and their display values:

<table>
<thead>
<tr>
<th>Entry</th>
<th>Value displayed</th>
</tr>
</thead>
<tbody>
<tr>
<td>“1947”</td>
<td>Jan 1 1947 12:00AM</td>
</tr>
<tr>
<td>“450128 12:30:1PM”</td>
<td>Jan 28 2045 12:30PM</td>
</tr>
<tr>
<td>“12:30.1PM 450128”</td>
<td>Jan 28 2045 12:30PM</td>
</tr>
<tr>
<td>“14:30.22”</td>
<td>Jan 1 1900 2:30PM</td>
</tr>
<tr>
<td>“4am”</td>
<td>Jan 1 1900 4:00AM</td>
</tr>
</tbody>
</table>

**Examples of date**

<table>
<thead>
<tr>
<th>Entry</th>
<th>Value displayed</th>
</tr>
</thead>
<tbody>
<tr>
<td>“1947”</td>
<td>Jan 1 1947</td>
</tr>
<tr>
<td>“450128”</td>
<td>Jan 28 2045</td>
</tr>
<tr>
<td>“520317”</td>
<td>Mar 17 1952</td>
</tr>
</tbody>
</table>

For bigdatetime and bigtime the value displays reflects a microsecond value. bigdatetime and bigtime have default display formats that accommodate their increased precision.

- hh:mm:ss.zzzzzzAM or PM

---

Table 1-15: Examples of datetime and date entries
The format for time must be specified as:

- `hours[:minutes[:seconds[:milliseconds]] [AM | PM]`
- `hours[:minutes[:seconds[number of milliseconds]] [AM | PM]`

Use 12 AM for midnight and 12 PM for noon. A bigtime value must contain either a colon or an AM or PM signifier. AM or PM can be entered in uppercase, lowercase, or mixed case.

The seconds specification can include either a decimal portion preceded by a point or a number of milliseconds preceded by a colon. For example, “12:30:20:1” means twenty seconds and one millisecond past 12:30; “12:30:20.1” means twenty and one-tenth of a second past.

To store a bigdatetime or bigtime value that includes microseconds, specify a string literal using a point. “00:00:00.1” means one tenth of a second past midnight and “00:00:00.000001” means one millionth of a second past midnight. Any value after the colon specifying fractional seconds will continue to refer to a number of milliseconds. Such as “00:00:00:5” means 5 milliseconds.

The display format for time values is “hh:mm:ss:mmmAM” (or “PM”); for example, “10:23:40:022PM”.

### Table 1-16: Examples of time entries

<table>
<thead>
<tr>
<th>Entry</th>
<th>Value displayed</th>
</tr>
</thead>
<tbody>
<tr>
<td>12:12:00&quot;</td>
<td>12:12PM</td>
</tr>
<tr>
<td>“01:23PM” or “01:23:1PM”</td>
<td>1:23PM</td>
</tr>
<tr>
<td>“02:24:00:001”</td>
<td>2:24AM</td>
</tr>
</tbody>
</table>

Use the `like` keyword to look for dates that match a particular pattern. If you use the equality operator (=) to search date or time values for a particular month, day, and year, Adaptive Server returns only those values for which the time is precisely 12:00:00:000AM.
Date and time datatypes

For example, if you insert the value “9:20” into a column named `arrival_time`, Adaptive Server converts the entry into “Jan 1 1900 9:20AM.” If you look for this entry using the equality operator, it is not found:

```
where arrival_time = "9:20" /* does not match */
```

You can find the entry using the `like` operator:

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

When using `like`, Adaptive Server first converts the dates to `datetime` or `date` format and then to `varchar`. The display format consists of the 3-character month in the current language, 2 characters for the day, 4 characters for the year, the time in hours and minutes, and “AM” or “PM.”

When searching with `like`, you cannot use the wide variety of input formats that are available for entering the date portion of `datetime`, `smalldatetime`, `bigdatetime`, `bigintime`, `date`, and `time` values. You cannot search for seconds or milliseconds with `like` and match a pattern, unless you are also using `style` 9 or 109 and the `convert` function.

If you are using `like`, and the day of the month is a number between 1 and 9, insert 2 spaces between the month and the day to match the `varchar` conversion of the `datetime` value. Similarly, if the hour is less than 10, the conversion places 2 spaces between the year and the hour. The following clause with 1 space between “May” and “2”) finds all dates from May 20 through May 29, but not May 2:

```
like "May 2%"
```

You do not need to insert the extra space with other date comparisons, only with `like`, since the `datetime` values are converted to `varchar` only for the `like` comparison.

Manipulating dates

You can do some arithmetic calculations on `date` and `time` datatypes values with the built-in date functions. See “Date functions” on page 73.

Standards and compliance

ANSI SQL – Compliance level: The `datetime` and `smalldatetime` datatypes are Transact-SQL extensions. `date` and `time` datatypes are entry-level compliant.
Character datatypes

Which datatype you use for a situation depends on the type of data you are storing:

- Use the character datatypes to store strings consisting of letters, numbers, and symbols.
- Use varchar(n) and char(n) for both single-byte character sets such as us_english and for multibyte character sets such as Japanese.
- Use the unichar(n) and univarchar(n) datatypes to store Unicode characters. They are useful for single-byte or multibyte characters when you need a fixed number of bytes per character.
- Use the fixed-length datatype, nchar(n), and the variable-length datatype, nvarchar(n), for both single-byte and multibyte character sets, such as Japanese. The difference between nchar(n) and char(n) and nvarchar(n) and varchar(n) is that both nchar(n) and nvarchar(n) allocate storage based on \( n \) times the number of bytes per character (based on the default character set), char(n) and varchar(n) allocate \( n \) bytes of storage.
- Character datatypes can store a maximum of a page size worth of data
- Use the text datatype (described in “text, image, and unitext datatypes” on page 36)—or multiple rows in a subtable—for strings longer than the char or varchar datatype allow.

**unichar, univarchar**

You can use the unichar and univarchar datatypes anywhere that you can use char and varchar character datatypes, without having to make syntax changes.

In Adaptive Server version 12.5.1 and later, queries containing character literals that cannot be represented in the server’s character set are automatically promoted to the unichar datatype so you do not have to make syntax changes for data manipulation language (DML) statements. Additional syntax is available for specifying arbitrary characters in character literals, but the decision to “promote” a literal to unichar is based solely on representability.
Character datatypes

With data definition language (DDL) statements, the syntax changes required are minimal. For example, in the create table command, the size of a Unicode column is specified in units of 16-bit Unicode values, not bytes, thereby maintaining the similarity between `char(200)` and `unichar(200)`. `sp_help`, which reports on the lengths of columns, uses the same units. The multiplication factor (2) is stored in the new global variable `@@unicharsize`.


Length and storage size

Character variables strip the trailing spaces from strings when the variable is populated in a `varchar` column of a cursor.

Use $n$ to specify the number of bytes of storage for `char` and `varchar` datatypes. For `unichar`, use $n$ to specify the number of Unicode characters (the amount of storage allocated is 2 bytes per character). For `nchar` and `nvarchar`, $n$ is the number of characters (the amount of storage allocated is $n$ times the number of bytes per character for the server’s current default character set).

If you do not use $n$ to specify the length:

- The default length is 1 byte for columns created with `create table`, `alter table`, and variables created with `declare`.
- The default length is 30 bytes for values created with the `convert` function.

Entries shorter than the assigned length are blank-padded; entries longer than the assigned length are truncated without warning, unless the `string_rtruncation` option to the `set` command is set to `on`. Fixed-length columns that allow nulls are internally converted to variable-length columns.

Use $n$ to specify the maximum length in characters for the variable-length datatypes, `varchar(n)`, `univarchar(n)`, and `nvarchar(n)`. Data in variable-length columns is stripped of trailing blanks; storage size is the actual length of the data entered. Data in variable-length variables and parameters retains all trailing blanks, but is not padded to the defined length. Character literals are treated as variable-length datatypes.

Fixed-length columns tend to take more storage space than variable-length columns, but are accessed somewhat faster. Table 1-17 summarizes the storage requirements of the different character datatypes:
Table 1-17: Character datatypes

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Stores</th>
<th>Bytes of storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>char(n)</td>
<td>Character</td>
<td>(n)</td>
</tr>
<tr>
<td>unichar(n)</td>
<td>Unicode character</td>
<td>(n \times @\text{unicharsize} \quad (@\text{unicharsize} \text{equals} \ 2))</td>
</tr>
<tr>
<td>nchar(n)</td>
<td>National character</td>
<td>(n \times @\text{ncharsize})</td>
</tr>
<tr>
<td>varchar(n)</td>
<td>Character varying</td>
<td>Actual number of characters entered</td>
</tr>
<tr>
<td>univarchar(n)</td>
<td>Unicode character varying</td>
<td>Actual number of characters (\times @\text{unicharsize})</td>
</tr>
<tr>
<td>nvarchar(n)</td>
<td>National character varying</td>
<td>Actual number of characters (\times @\text{ncharsize})</td>
</tr>
</tbody>
</table>

Determining column length with system functions

Use the `char_length` string function and `datalength` system function to determine column length:

- `char_length` returns the number of characters in the column, stripping trailing blanks for variable-length datatypes.
- `datalength` returns the number of bytes, stripping trailing blanks for data stored in variable-length columns.

When a `char` value is declared to allow NULL values, Adaptive Server stores it internally as a `varchar`.

If the `min` or `max` aggregate functions are used on a `char` column, the result returned is `varchar`, and is therefore stripped of all trailing spaces.

Entering character data

Character strings must be enclosed in single or double quotes. If you use `set quoted_identifier` on, use single quotes for character strings; otherwise, Adaptive Server treats them as identifiers.

Strings that include the double-quote character should be surrounded by single quotes. Strings that include the single-quote character should be surrounded by double quotes. For example:

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

An alternative is to enter two quotation marks for each quotation mark you want to include in the string. For example:

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

To continue a character string onto the next line of your screen, enter a backslash (\) before going to the next line.
Character datatypes

For more information about quoted identifiers, see the section “Delimited identifiers” of the Transact SQL User’s Guide.

Entering Unicode characters

Optional syntax allows you to specify arbitrary Unicode characters. If a character literal is immediately preceded by U& or u& (with no intervening white space), the parser recognizes escape sequences within the literal. An escape sequence of the form \xxxx (where xxxx represents four hexadecimal digits) is replaced with the Unicode character whose scalar value is xxxx. Similarly, an escape sequence of the form \+yyyyyy is replaced with the Unicode character whose scalar value is yyyyyy. The escape sequence \ is replaced by a single \. For example, the following is equivalent to:

```sql
select * from mytable where unichar_column = '𠮷'
```

select * from mytable where unichar_column = U'𠮷'

The U& or u& prefix simply enables the recognition of escapes. The datatype of the literal is chosen solely on the basis of representability. Thus, for example, the following two queries are equivalent:

```sql
select * from mytable where char_column = 'A'
selct * from mytable where char_column = U'\0041'
```

In both cases, the datatype of the character literal is char, since “A” is an ASCII character, and ASCII is a subset of all Sybase-supported server character sets.

The U& and u& prefixes also work with the double-quoted character literals and for quoted identifiers. However, quoted identifiers must be representable in the server’s character set, insofar as all database objects are identified by names in system tables, and all such names are of datatype char.

Treatment of blanks

The following example creates a table named spaces that has both fixed- and variable-length character columns:

```sql
create table spaces (cnot char(5) not null,
cnull char(5) null,
vnot varchar(5) not null,
vnull varchar(5) null,
explanation varchar(25) not null)
```

```sql
select * from mytable where unichar_column = '    '
```
CHAPTER 1  System and User-Defined Datatypes

This example illustrates how the column’s datatype and null type interact to determine how blank spaces are treated:

- Only char not null and nchar not null columns are padded to the full width of the column; char null columns are treated like varchar and nchar null columns are treated like nvarchar.
- Only unichar not null columns are padded to the full width of the column; unichar null columns are treated like univarchar.
- Preceding blanks are not affected.
- Trailing blanks are truncated except for char, unichar, and nchar not null columns.
- The empty string (""") is treated as a single space. In char, nchar, and unichar not null columns, the result is a column-length field of spaces.

Reference Manual: Building Blocks 31
Manipulating character data

You can use the like keyword to search character strings for particular characters and the built-in string functions to manipulate their contents. You can use strings consisting of numbers for arithmetic after being converted to exact and approximate numeric datatypes with the convert function.

Standards and compliance

ANSI SQL – Compliance level: Transact-SQL provides the char and varchar ANSI SQL datatypes. The nchar, nvarchar, unichar, and univarchar datatypes are Transact-SQL extensions.

Binary datatypes

Use the binary datatypes, binary(n) and varbinary(n), to store raw binary data, such as pictures, in a raw binary notation, up to the maximum column size for your server’s logical page size.

Valid binary and varbinary entries

Binary data begins with the characters “0x” and can include any combination of digits, and the uppercase and lowercase letters A through F.

Use n to specify the column length in bytes, or use the default length of 1 byte. Each byte stores 2 binary digits. If you enter a value longer than n, Adaptive Server truncates the entry to the specified length without warning or error.

Use the fixed-length binary type, binary(n), for data in which all entries are expected to be approximately equal in length.

Use the variable-length binary type, varbinary(n), for data that is expected to vary greatly in length.

Because entries in binary columns are zero-padded to the column length (n), they may require more storage space than those in varbinary columns, but they are accessed somewhat faster.

If you do not use n to specify the length:
The default length is 1 byte for columns created with create table, alter table, and variables created with declare.

The default length is 30 bytes for values created with the convert function.

Entries of more than the maximum column size

Use the image datatype to store larger blocks of binary data (up to 2,147,483,647 bytes) on external data pages. You cannot use the image datatype for variables or for parameters in stored procedures. For more information, see “text, image, and unitext datatypes” on page 36.

Treatment of trailing zeros

All binary not null columns are padded with zeros to the full width of the column. Trailing zeros are truncated in all varbinary data and in binary null columns, since columns that accept null values must be treated as variable-length columns.

The following example creates a table with all four variations of binary and varbinary datatypes, NULL, and NOT NULL. The same data is inserted in all four columns and is padded or truncated according to the datatype of the column.

```
create table zeros (bnot binary(5) not null,
                  bnull binary(5) null,
                  vnot varbinary(5) not null,
                  vnull varbinary(5) null)
insert zeros values (0x12345000, 0x12345000, 0x12345000, 0x12345000)
insert zeros values (0x123, 0x123, 0x123, 0x123)
select * from zeros
```

<table>
<thead>
<tr>
<th>bnot</th>
<th>bnull</th>
<th>vnot</th>
<th>vnull</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1234500000</td>
<td>0x123450</td>
<td>0x123450</td>
<td>0x123450</td>
</tr>
<tr>
<td>0x0123000000</td>
<td>0x0123</td>
<td>0x0123</td>
<td>0x0123</td>
</tr>
</tbody>
</table>
Because each byte of storage holds 2 binary digits, Adaptive Server expects binary entries to consist of the characters “0x” followed by an even number of digits. When the “0x” is followed by an odd number of digits, Adaptive Server assumes that you omitted the leading 0 and adds it for you.

Input values “0x00” and “0x0” are stored as “0x00” in variable-length binary columns (binary null, image, and varbinary columns). In fixed-length binary (binary not null) columns, the value is padded with zeros to the full length of the field:

```
insert zeros values (0x0, 0x0,0x0, 0x0)
select * from zeros where bnot = 0x00
```

```
<table>
<thead>
<tr>
<th>bnot</th>
<th>bnull</th>
<th>vnot</th>
<th>vnull</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000000000</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
</tr>
</tbody>
</table>
```

If the input value does not include the “0x”, Adaptive Server assumes that the value is an ASCII value and converts it. For example:

```
create table sample (col_a binary(8))
insert sample values ('002710000000ae1b')
select * from sample
```

```
col_a
------------------
0x3030323731303030
```

**Platform dependence**

The exact form in which you enter a particular value depends upon the platform you are using. Therefore, calculations involving binary data can produce different results on different machines.

You cannot use the aggregate functions sum or avg with the binary datatypes.

For platform-independent conversions between hexadecimal strings and integers, use the inttoph and hextoint functions rather than the platform-specific convert function. For details, see “Datatype conversion functions” on page 63.
Standards and compliance

ANSI SQL – Compliance level: The binary and varbinary datatypes are Transact-SQL extensions.

**bit datatype**

Use the bit datatype for columns that contain true/false and yes/no types of data. The status column in the syscolumns system table indicates the unique offset position for bit datatype columns.

bit columns hold either 0 or 1. Integer values other than 0 or 1 are accepted, but are always interpreted as 1.

Storage size is 1 byte. Multiple bit datatypes in a table are collected into bytes. For example, 7 bit columns fit into 1 byte; 9 bit columns take 2 bytes.

Columns with a datatype of bit cannot be NULL and cannot have indexes on them.

Standards and compliance

ANSI SQL – Compliance level: Transact-SQL extension.

**sysname and longsysname datatypes**

sysname and longsysname are user-defined datatypes that are distributed on the Adaptive Server installation tape and used in the system tables. The definitions are:

- sysname – varchar(30) "not null"
- longsysname – varchar(255) "not null"

You can declare a column, parameter, or variable to be of types sysname and longsysname. Alternately, you can also create a user-defined datatype with a base type of sysname and longsysname, and then define columns, parameters, and variables with the user-defined datatype.
Standards and compliance

ANSI SQL – Compliance level: All user-defined datatypes, including sysname and longsysname, are Transact-SQL extensions.

text, image, and unitext datatypes

text columns are variable-length columns that can hold up to 2,147,483,647 (2^{31} - 1) bytes of printable characters.

The variable-length unitext datatype can hold up to 1,073,741,823 Unicode characters (2,147,483,646 bytes).

image columns are variable-length columns that can hold up to 2,147,483,647 (2^{31} - 1) bytes of raw binary data.

A key distinction between text and image is that text is subject to character-set conversion if you are not using the default character set of Adaptive Server default. image is not subject to character-set conversion.

Define a text, unitext, or image column as you would any other column, with a create table or alter table statement. text, unitext, or image datatype definitions do not include lengths. text, unitext, and image columns do permit null values. Their column definition takes the form:

```
column_name (text | image | unitext) [null]
```

For example, the create table statement for the author’s blurbs table in the pubs2 database with a text column, blurb, that permits null values, is:

```
create table blurbs
(au_id id not null,
 copy text null)
```

This example creates a unitext column that allows null values:

```
create table tb (ut unitext null)
```

To create the au_pix table in the pubs2 database with an image column:

```
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)
```
Adaptive Server stores text, unitext, and image data in a linked list of data pages that are separate from the rest of the table. Each text, unitext, or image page stores one logical page size worth of data (2, 4, 8, or 16K). All text, unitext, and image data for a table is stored in a single page chain, regardless of the number of text, unitext, and image columns the table contains.

You can place subsequent allocations for text, unitext, and image data pages on a different logical device with sp_placeobject.

Image values that have an odd number of hexadecimal digits are padded with a leading zero (an insert of “0xaaabb” becomes “0x0aaabb”).

You can use the partition option of the alter table command to partition a table that contains text, unitext, and image columns. Partitioning the table creates additional page chains for the other columns in the table, but has no effect on the way the text, unitext, and image columns are stored.

You can use unitext anywhere you use the text datatype, with the same semantics. Unitext columns are stored in UTF-16 encoding, regardless of the Adaptive Server default character set.

Data structures used for storing text, unitext, and image data

When you allocate text, unitext, or image data, a 16-byte text pointer is inserted into the row you allocated. Part of this text pointer refers to a text page number at the head of the text, unitext, or image data. This text pointer is known as the first text page.

The first text page contains two parts:

- The text data page chain, which contains the text and image data and is a double-linked list of text pages
- The optional text-node structure, which is used to access the user text data

Once an first text page is allocated for text, unitext, or image data, it is never deallocated. If an update to an existing text, unitext, or image data row results in fewer text pages than are currently allocated for this text, unitext, or image data, Adaptive Server deallocates the extra text pages. If an update to text, unitext, or image data sets the value to NULL, all pages except the first text page are deallocated.

Figure 1-1 shows the relationship between the data row and the text pages.
Initializing **text**, **unitext**, and **image** columns

text, unitext, and image columns are not initialized until you update them or insert a non-null value. Initialization allocates at least one data page for each non-null text, unitext, or image data value. It also creates a pointer in the table to the location of the text, unitext, or image data.

For example, the following statements create the table `texttest` and initialize the `blurb` column by inserting a non-null value. The column now has a valid text pointer, and the first text page has been allocated.

```sql
create table texttest
    (title_id varchar(6), blurb text null, pub_id char(4))
insert texttest values
    ("BU7832", "Straight Talk About Computers is an annotated analysis of what computers can do for you: a no-hype guide for the critical user.", "1389")
```

The following statements create a table for image values and initialize the image column:

```sql
create table imagetest
    (image_id varchar(6), imagecol image null, graphic_id char(4))
```
insert imagetest values
("94732", 0x0000008300000000000100000000013c, "1389")

Note Surround text values with quotation marks and precede image values with the characters “0x”.

For information on inserting and updating text, unitext, and image data with Client-Library programs, see the Client-Library/C Reference Manual.

Defining unitext columns

You can define a unitext column the same way you define other datatypes, using create table or alter table statements. You do not define the length of a unitext column, and the column can be null.

This example creates a unitext column that allows null values:

create table tb (ut unitext null)

default unicode sort order defines the sort order for unitext columns for pattern matching in like clauses and in the patindex function, this is independent of the Adaptive Server default sort order.

Saving space by allowing NULL

To save storage space for empty text, unitext, or image columns, define them to permit null values and insert nulls until you use the column. Inserting a null value does not initialize a text, unitext, or image column and, therefore, does not create a text pointer or allocate storage. For example, the following statement inserts values into the title_id and pub_id columns of the testtext table created above, but does not initialize the blurb text column:

insert texttest
(title_id, pub_id) values ("BU7832", "1389")

Getting information from sysindexes

Each table with text, unitext, or image columns has an additional row in sysindexes that provides information about these columns. The name column in sysindexes uses the form “tablename.” The indid is always 255. These columns provide information about text storage:
text, image, and unitext datatypes

Table 1-18: Storage of text and image data

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ioampg</td>
<td>Pointer to the allocation page for the text page chain</td>
</tr>
<tr>
<td>first</td>
<td>Pointer to the first page of text data</td>
</tr>
<tr>
<td>root</td>
<td>Pointer to the last page</td>
</tr>
<tr>
<td>segment</td>
<td>Number of the segment where the object resides</td>
</tr>
</tbody>
</table>

You can query the sysindexes table for information about these columns. For example, the following query reports the number of data pages used by the blursbs table in the pubs2 database:

```sql
select name, data_pages(db_id(), object_id("blursbs"), indid)
from sysindexes
where name = "tblursbs"
```

**Note** The system tables poster shows a one-to-one relationship between sysindexes and systabstats. This is correct, except for text and image columns, for which information is not kept in systabstats.

Using `readtext` and `writetext`

Before you can use `writetext` to enter text data or `readtext` to read it, you must initialize the text column. For details, see `readtext` and `writetext` in *Reference Manual: Commands*.

Using `update` to replace existing text, unitext, and image data with NULL reclaims all allocated data pages except the first page, which remains available for future use of `writetext`. To deallocate all storage for the row, use `delete` to remove the entire row.

There are restrictions for using `readtext` and `writetext` on a column defined for unitext. For more information see the “Usage” sections under `readtext` and `writetext` in the *Reference Manual: Commands*.

Determining how much space a column uses

`sp_spaceused` provides information about the space used for text data as `index_size`:

```sql
sp_spaceused blursbs
```
Restrictions on text, image, and unitext columns

You cannot use text, image, or unitext columns:

- As parameters to stored procedures or as values passed to these parameters
- As local variables
- In order by clause, compute clause, group by, and union clauses
- In an index
- In subqueries or joins
- In a where clause, except with the keyword like
- With the + concatenation operator

Selecting text, unitext, and image data

The following global variables return information on text, unitext, and image data:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>@@textptr</td>
<td>The text pointer of the last text, unitext, or image column inserted or updated by a process. Do not confuse this global variable with the textptr function.</td>
</tr>
<tr>
<td>@@textcolid</td>
<td>ID of the column referenced by @@textptr.</td>
</tr>
<tr>
<td>@@textdbid</td>
<td>ID of a database containing the object with the column referenced by @@textptr.</td>
</tr>
<tr>
<td>@@textobjid</td>
<td>ID of the object containing the column referenced by @@textptr.</td>
</tr>
<tr>
<td>@@textsiz</td>
<td>Current value of the set textsize option, which specifies the maximum length, in bytes, of text, unitext, or image data to be returned with a select statement. It defaults to 32K. The maximum size for @@textsize is $2^{31} - 1$ (that is, 2,147,483,647).</td>
</tr>
<tr>
<td>@@textts</td>
<td>Text timestamp of the column referenced by @@textptr.</td>
</tr>
</tbody>
</table>
Converting text and image datatypes

You can explicitly convert text values to char, unichar, varchar, and univarchar, and image values to binary or varbinary with the convert function, but you are limited to the maximum length of the character and binary datatypes, which is determined by the maximum column size for your server’s logical page size. If you do not specify the length, the converted value has a default length of 30 bytes. Implicit conversion is not supported.

Converting to or from unitext

You can implicitly convert any character or binary datatype to unitext, as well as explicitly convert to and from unitext to other datatypes. The conversion result, however, is limited to the maximum length of the destination datatype. When a unitext value cannot fit the destination buffer on a Unicode character boundary, data is truncated. If you have enabled enable surrogate processing, the unitext value is never truncated in the middle of a surrogate pair of values, which means that fewer bytes may be returned after the datatype conversion.

For example, if a unitext column ut in table tb stores the string “U+0041U+0042U+00c2” (U+0041 representing the Unicode character “A”), this query returns the value “AB” if the server’s character set is UTF-8, because U+00C2 is converted to 2-byte UTF-8 0xc382:

```sql
select convert(char(3), ut) from tb
```

Table 1-20: Converting to and from unitext

<table>
<thead>
<tr>
<th>These datatypes convert implicitly to unitext</th>
<th>These datatypes convert implicitly from unitext</th>
<th>These datatypes convert explicitly from unitext</th>
</tr>
</thead>
<tbody>
<tr>
<td>char, varchar, unichar, univarchar, binary, varbinary, text, image</td>
<td>text, image</td>
<td>char, varchar, unichar, univarchar, binary, varbinary</td>
</tr>
</tbody>
</table>

The alter table modify command does not support text, image, or unitext columns to be the modified column. To migrate from a text to a unitext column:

- Use bcp out -Jutf8 out to copy text column data out
- Create a table with unitext columns
- Use bcp in -Jutf8 to insert data into the new table
Pattern matching in text data

Use the `patindex` function to search for the starting position of the first occurrence of a specified pattern in a text, unitext, varchar, univarchar, unichar, or char column. The `%` wildcard character must precede and follow the pattern (except when you are searching for the first or last character).

You can also use the `like` keyword to search for a particular pattern. The following example selects each text data value from the `copy` column of the `blurbs` table that contains the pattern “Net Etiquette.”

```sql
select copy from blurbs
where copy like "%Net Etiquette%"
```

Duplicate rows

The pointer to the text, image, and unitext data uniquely identifies each row. Therefore, a table that contains text, image, and unitext data does not contain duplicate rows unless there are rows in which all text, image, and unitext data is NULL. If this is the case, the pointer has not been initialized.

Standards and compliance

ANSI SQL – Compliance level: The text, image, and unitext datatypes are Transact-SQL extensions.

Datatypes and encrypted columns

Table 1-21 lists the supported datatypes for encrypted columns, as well as the on-disk length of encrypted columns for datatypes supported for Adaptive Server version 15.0.2.

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Input data length</th>
<th>Encrypted column type</th>
<th>Max encrypted data length (no init_vector)</th>
<th>Actual encrypted data length (no init_vector)</th>
<th>Max encrypted data length with init_vector</th>
<th>Actual encrypted data length with init_vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>4</td>
<td>varbinary</td>
<td>17</td>
<td>17</td>
<td>33</td>
<td>33</td>
</tr>
</tbody>
</table>
User-defined datatypes

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Input data length</th>
<th>Encrypted column type</th>
<th>Max encrypted data length (no init_vector)</th>
<th>Actual encrypted data length (no init_vector)</th>
<th>Max encrypted data length with init_vector</th>
<th>Actual encrypted data length (with init_vector)</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>4</td>
<td>varbinary</td>
<td>17</td>
<td>17</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>smalldatetime</td>
<td>4</td>
<td>varbinary</td>
<td>17</td>
<td>17</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>bigdatetime</td>
<td>8</td>
<td>varbinary</td>
<td>17</td>
<td>17</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>bigtime</td>
<td>8</td>
<td>varbinary</td>
<td>17</td>
<td>17</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>datetime</td>
<td>8</td>
<td>varbinary</td>
<td>17</td>
<td>17</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>smallmoney</td>
<td>4</td>
<td>varbinary</td>
<td>17</td>
<td>17</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>money</td>
<td>8</td>
<td>varbinary</td>
<td>17</td>
<td>17</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>bit</td>
<td>8</td>
<td>varbinary</td>
<td>17</td>
<td>17</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>bigint</td>
<td>8</td>
<td>varbinary</td>
<td>17</td>
<td>17</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>unsigned bigint</td>
<td>8</td>
<td>varbinary</td>
<td>17</td>
<td>17</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>unichar(10)</td>
<td>2 (1 unichar character)</td>
<td>varbinary</td>
<td>33</td>
<td>17</td>
<td>49</td>
<td>33</td>
</tr>
<tr>
<td>unichar(10)</td>
<td>20 (10 unichar characters)</td>
<td>varbinary</td>
<td>33</td>
<td>33</td>
<td>49</td>
<td>49</td>
</tr>
<tr>
<td>univarchar(20)</td>
<td>20 (10 unichar characters)</td>
<td>varbinary</td>
<td>49</td>
<td>33</td>
<td>65</td>
<td>49</td>
</tr>
</tbody>
</table>

text, image, and unitext datatypes are not supported for this release of Adaptive Server.

User-defined datatypes

User-defined datatypes are built from the system datatypes and from the sysname or longsysname user-defined datatypes. After you create a user-defined datatype, you can use it to define columns, parameters, and variables. Objects that are created from user-defined datatypes inherit the rules, defaults, null type, and IDENTITY property of the user-defined datatype, as well as inheriting the defaults and null type of the system datatypes on which the user-defined datatype is based.
A user-defined datatype must be created in each database in which it will be used. Create frequently used types in the model database. These types are automatically added to each new database (including tempdb, which is used for temporary tables) as it is created.

Adaptive Server allows you to create user-defined datatypes, based on any system datatype, using sp_addtype. You cannot create a user-defined datatype based on another user-defined datatype, such as timestamp or the tid datatype in the pubs2 database.

The sysname and longsysname datatypes are exceptions to this rule. Though sysname and longsysname are user-defined datatypes, you can use them to build user-defined datatypes.

User-defined datatypes are database objects. Their names are case-sensitive and must conform to the rules for identifiers.

You can bind rules to user-defined datatypes with sp_bindrule and bind defaults with sp_bindefault.

By default, objects built on a user-defined datatype inherit the user-defined datatype’s null type or IDENTITY property. You can override the null type or IDENTITY property in a column definition.

Use sp_rename to rename a user-defined datatype.

Use sp_droptype to remove a user-defined datatype from a database.

**Note** You cannot drop a datatype that is already in use in a table.

Use sp_help to display information about the properties of a system datatype or a user-defined datatype. You can also use sp_help to display the datatype, length, precision, and scale for each column in a table.

**Standards and compliance**

ANSI SQL – Compliance level: User-defined datatypes are a Transact-SQL extension.
User-defined datatypes
CHAPTER 2

Transact-SQL Functions

This chapter describes the Transact-SQL functions. Functions are used to return information from the database. They are allowed in the select list, in the where clause, and anywhere an expression is allowed. They are often used as part of a stored procedure or program.

Types of functions

Table 2-1 lists the different types of Transact-SQL functions and describes the type of information each returns.

<table>
<thead>
<tr>
<th>Topics</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Types of functions</td>
<td>47</td>
</tr>
<tr>
<td>Aggregate functions</td>
<td>54</td>
</tr>
<tr>
<td>Statistical aggregate functions</td>
<td>60</td>
</tr>
<tr>
<td>Datatype conversion functions</td>
<td>63</td>
</tr>
<tr>
<td>Date functions</td>
<td>73</td>
</tr>
<tr>
<td>Mathematical functions</td>
<td>74</td>
</tr>
<tr>
<td>Security functions</td>
<td>75</td>
</tr>
<tr>
<td>String functions</td>
<td>76</td>
</tr>
<tr>
<td>System functions</td>
<td>77</td>
</tr>
<tr>
<td>Text and image functions</td>
<td>78</td>
</tr>
<tr>
<td>User-defined SQL functions</td>
<td>78</td>
</tr>
</tbody>
</table>

Table 2-1: Types of Transact-SQL functions

<table>
<thead>
<tr>
<th>Type of function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregate functions</td>
<td>Generate summary values that appear as new columns or as additional rows in the query results.</td>
</tr>
<tr>
<td>Datatype conversion functions</td>
<td>Change expressions from one datatype to another and specify new display formats for date and time information.</td>
</tr>
<tr>
<td>Date functions</td>
<td>Perform computations on datetime, smalldatetime, date, and time values and their components, date parts.</td>
</tr>
<tr>
<td>Mathematical functions</td>
<td>Commonly needed for operations on mathematical data.</td>
</tr>
</tbody>
</table>
## Types of functions

<table>
<thead>
<tr>
<th>Type of function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String functions</td>
<td>Operate on binary data, character strings, and expressions.</td>
</tr>
<tr>
<td>System functions</td>
<td>Retrieves special information from the database and database objects.</td>
</tr>
<tr>
<td>Text and image functions</td>
<td>Supply values commonly needed for operations on text, unitext, and image data.</td>
</tr>
</tbody>
</table>

Table 2-2 lists the functions in alphabetical order.

<table>
<thead>
<tr>
<th>Function</th>
<th>Type</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>abs</code> on page 80</td>
<td>Mathematical</td>
<td>The absolute value of an expression.</td>
</tr>
<tr>
<td><code>acos</code> on page 81</td>
<td>Mathematical</td>
<td>The angle (in radians) with a specified cosine.</td>
</tr>
<tr>
<td><code>ascii</code> on page 82</td>
<td>String</td>
<td>The ASCII code for the first character in an expression.</td>
</tr>
<tr>
<td><code>asin</code> on page 84</td>
<td>Mathematical</td>
<td>The angle (in radians) with a specified sine.</td>
</tr>
<tr>
<td><code>atan</code> on page 85</td>
<td>Mathematical</td>
<td>The angle (in radians) with a specified tangent.</td>
</tr>
<tr>
<td><code>atn2</code> on page 86</td>
<td>Mathematical</td>
<td>The angle (in radians) with specified sine and cosine.</td>
</tr>
<tr>
<td><code>audit_event_name</code></td>
<td>Security</td>
<td>A description of an audit event</td>
</tr>
<tr>
<td><code>avg</code> on page 87</td>
<td>Aggregate</td>
<td>The numeric average of all (distinct) values.</td>
</tr>
<tr>
<td><code>biginttohex</code> on page 92</td>
<td>Datatype conversion</td>
<td>Returns the platform-independent hexadecimal equivalent of the specified integer.</td>
</tr>
<tr>
<td><code>case</code> on page 96</td>
<td>Conversion</td>
<td>Allows SQL expressions to be written for conditional values. case expressions can be used anywhere a value expression can be used.</td>
</tr>
<tr>
<td><code>cast</code> on page 99</td>
<td>Datatype</td>
<td>A specified value, converted to another datatype</td>
</tr>
<tr>
<td><code>ceiling</code> on page 102</td>
<td>Mathematical</td>
<td>The smallest integer greater than or equal to the specified value.</td>
</tr>
<tr>
<td><code>char</code> on page 104</td>
<td>String</td>
<td>The character equivalent of an integer.</td>
</tr>
<tr>
<td><code>charindex</code> on page 108</td>
<td>String</td>
<td>Returns an integer representing the starting position of an expression.</td>
</tr>
<tr>
<td><code>char_length</code> on page 106</td>
<td>String</td>
<td>The number of characters in an expression.</td>
</tr>
<tr>
<td><code>col_length</code> on page 111</td>
<td>System</td>
<td>The defined length of a column.</td>
</tr>
<tr>
<td><code>col_name</code> on page 112</td>
<td>System</td>
<td>The name of the column with specified table and column IDs.</td>
</tr>
<tr>
<td><code>compare</code> on page 113</td>
<td>System</td>
<td>Returns the following values, based on the collation rules that you chose:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1 – indicates that <code>char_expression1</code> is greater than <code>char_expression2</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 0 – indicates that <code>char_expression1</code> is equal to <code>char_expression2</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• -1 – indicates that <code>char_expression1</code> is less than <code>char_expression2</code></td>
</tr>
<tr>
<td><code>convert</code> on page 118</td>
<td>Datatype conversion</td>
<td>The specified value, converted to another datatype or a different datetime display format.</td>
</tr>
</tbody>
</table>
### Function Type Return value

<table>
<thead>
<tr>
<th>Function</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>cos</strong> on page 124</td>
<td>Mathematical</td>
<td>The cosine of the specified angle (in radians).</td>
</tr>
<tr>
<td><strong>cot</strong> on page 125</td>
<td>Mathematical</td>
<td>The cotangent of the specified angle (in radians).</td>
</tr>
<tr>
<td><strong>count</strong> on page 126</td>
<td>Aggregate</td>
<td>The number of (distinct) non-null values as an integer.</td>
</tr>
<tr>
<td><strong>count_big</strong> on page 128</td>
<td>Aggregate</td>
<td>The number of (distinct) non-null values as a bigint.</td>
</tr>
<tr>
<td><strong>current_date</strong> on page 132</td>
<td>Date</td>
<td>Returns the current date.</td>
</tr>
<tr>
<td><strong>current_time</strong> on page 133</td>
<td>Date</td>
<td>Returns the current time.</td>
</tr>
<tr>
<td><strong>curunreservedpgs</strong> on page 134</td>
<td>System</td>
<td>The number of free pages in the specified disk piece.</td>
</tr>
<tr>
<td><strong>data_pages</strong> on page 136</td>
<td>System</td>
<td>The number of pages used by the specified table or index.</td>
</tr>
<tr>
<td><strong>datalength</strong> on page 140</td>
<td>System</td>
<td>The actual length, in bytes, of the specified column or string.</td>
</tr>
<tr>
<td><strong>dateadd</strong> on page 141</td>
<td>Date</td>
<td>The date produced by adding a given number of years, quarters, hours, or other date parts to the specified date.</td>
</tr>
<tr>
<td><strong>datediff</strong> on page 144</td>
<td>Date</td>
<td>The difference between two date expressions.</td>
</tr>
<tr>
<td><strong>datename</strong> on page 148</td>
<td>Date</td>
<td>The name of the specified part of a date expression.</td>
</tr>
<tr>
<td><strong>datepart</strong> on page 150</td>
<td>Date</td>
<td>The integer value of the specified part of a date expression.</td>
</tr>
<tr>
<td><strong>day</strong> on page 155</td>
<td>Date</td>
<td>Returns an integer that represents the day in the datepart of a specified date.</td>
</tr>
<tr>
<td><strong>db_id</strong> on page 158</td>
<td>System</td>
<td>The ID number of the specified database.</td>
</tr>
<tr>
<td><strong>db_name</strong> on page 160</td>
<td>System</td>
<td>The name of the database with a specified ID number.</td>
</tr>
<tr>
<td><strong>degrees</strong> on page 162</td>
<td>Mathematical</td>
<td>The size, in degrees, of an angle with a specified number of radians.</td>
</tr>
<tr>
<td><strong>derived_stat</strong> on page 163</td>
<td>System</td>
<td>Returns derived statistics for the specified object and index.</td>
</tr>
<tr>
<td><strong>difference</strong> on page 168</td>
<td>String</td>
<td>The difference between two soundex values.</td>
</tr>
<tr>
<td><strong>exp</strong> on page 169</td>
<td>Mathematical</td>
<td>The value that results from raising the constant e to the specified power.</td>
</tr>
<tr>
<td><strong>floor</strong> on page 170</td>
<td>Mathematical</td>
<td>The largest integer that is less than or equal to the specified value.</td>
</tr>
<tr>
<td><strong>get_appcontext</strong> on page 172</td>
<td>Security</td>
<td>Returns the value of the attribute in a specified context.</td>
</tr>
<tr>
<td><strong>getdate</strong> on page 173</td>
<td>Date</td>
<td>The current system date and time.</td>
</tr>
<tr>
<td><strong>hextobigint</strong> on page 181</td>
<td>Datatype conversion</td>
<td>The bigint value equivalent of a hexadecimal string</td>
</tr>
<tr>
<td><strong>hextoint</strong> on page 182</td>
<td>Datatype conversion</td>
<td>The platform-independent integer equivalent of the specified hexadecimal string.</td>
</tr>
<tr>
<td><strong>host_id</strong> on page 183</td>
<td>System</td>
<td>Returns the client computer’s operating system process ID for the current Adaptive Server client.</td>
</tr>
</tbody>
</table>
### Types of functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Type</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>host_name</td>
<td>System</td>
<td>The current host computer name of the client process.</td>
</tr>
<tr>
<td>identity_burn_max</td>
<td>System</td>
<td>The identity_burn_max value.</td>
</tr>
<tr>
<td>index_col</td>
<td>System</td>
<td>The name of the indexed column in the specified table or view.</td>
</tr>
<tr>
<td>index_colorder</td>
<td>System</td>
<td>Returns the column order</td>
</tr>
<tr>
<td>inttohex</td>
<td>Datatype conversion</td>
<td>The platform-independent, hexadecimal equivalent of the specified integer.</td>
</tr>
<tr>
<td>isdate</td>
<td>Datatype conversion</td>
<td>Determines whether an input expression is a valid datetime value</td>
</tr>
<tr>
<td>isnumeric</td>
<td>Datatype conversion</td>
<td>Determines if an expression is a valid numeric datatype</td>
</tr>
<tr>
<td>is_quiesced</td>
<td>System</td>
<td>Indicates whether a database is in quiesce database mode. is_quiesced returns 1 if the database is quiesced and 0 if it is not.</td>
</tr>
<tr>
<td>is_sec_service_on</td>
<td>Security</td>
<td>1 if the security service is active; 0 if it is not.</td>
</tr>
<tr>
<td>isnull</td>
<td>System</td>
<td>Substitutes the value specified in expression2 when expression1 evaluates to NULL.</td>
</tr>
<tr>
<td>lct_admin</td>
<td>System</td>
<td>Manages the last-chance threshold.</td>
</tr>
<tr>
<td>left</td>
<td>String</td>
<td>Returns a specified number of characters on the left end of a character string.</td>
</tr>
<tr>
<td>len</td>
<td>String</td>
<td>Returns the number of characters, not the number of bytes, of a specified string expression, excluding trailing blanks.</td>
</tr>
<tr>
<td>license_enabled</td>
<td>System</td>
<td>1” if the feature's license is enabled; 0 if it is not.</td>
</tr>
<tr>
<td>list_appcontext</td>
<td>Security</td>
<td>Lists all the attributes of all the contexts in the current session.</td>
</tr>
<tr>
<td>lockscheme</td>
<td>Mathematical</td>
<td>Returns the locking scheme of the specified object as a string.</td>
</tr>
<tr>
<td>log</td>
<td>Mathematical</td>
<td>The natural logarithm of the specified number.</td>
</tr>
<tr>
<td>log10</td>
<td>Mathematical</td>
<td>The base 10 logarithm of the specified number.</td>
</tr>
<tr>
<td>lower</td>
<td>String</td>
<td>The lowercase equivalent of the specified expression.</td>
</tr>
<tr>
<td>ltrim</td>
<td>String</td>
<td>The specified expression, trimmed of leading blanks</td>
</tr>
<tr>
<td>max</td>
<td>Aggregate</td>
<td>The highest value in a column.</td>
</tr>
<tr>
<td>min</td>
<td>Aggregate</td>
<td>The lowest value in a column.</td>
</tr>
<tr>
<td>month</td>
<td>Date</td>
<td>An integer that represents the month in the datepart of a specified date</td>
</tr>
<tr>
<td>mut_excl_roles</td>
<td>Security</td>
<td>The mutual exclusivity between two roles.</td>
</tr>
<tr>
<td>Function</td>
<td>Type</td>
<td>Return value</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>newid on page 218</td>
<td>System</td>
<td>Generates human-readable, globally unique IDs (GUIDs) in two different formats, based on arguments you provide.</td>
</tr>
<tr>
<td>next_identity on page 220</td>
<td>System</td>
<td>Retrieves the next identity value that is available for the next insert.</td>
</tr>
<tr>
<td>nullif on page 221</td>
<td>System</td>
<td>Allows SQL expressions to be written for conditional values. nullif expressions can be used anywhere a value expression can be used; alternative for a case expression.</td>
</tr>
<tr>
<td>object_id on page 227</td>
<td>System</td>
<td>The object ID of the specified object.</td>
</tr>
<tr>
<td>object_name on page 228</td>
<td>System</td>
<td>The name of the object with the specified object ID.</td>
</tr>
<tr>
<td>pagesize on page 230</td>
<td>Mathematical</td>
<td>Returns the page size, in bytes, for the specified object.</td>
</tr>
<tr>
<td>partition_id on page 232</td>
<td>System</td>
<td>Returns the partition ID of the specified data or index partition name.</td>
</tr>
<tr>
<td>partition_name on page 233</td>
<td>System</td>
<td>The explicit name of a new partition, partition_name returns the partition name of the specified data or index partition id.</td>
</tr>
<tr>
<td>partition_object_id on page 234</td>
<td>System</td>
<td>Displays the object ID for a specified partition ID and database ID.</td>
</tr>
<tr>
<td>patindex on page 235</td>
<td>String, Text, Unitext, and Image</td>
<td>The starting position of the first occurrence of a specified pattern.</td>
</tr>
<tr>
<td>pi on page 238</td>
<td>Mathematical</td>
<td>The constant value 3.1415926535897936.</td>
</tr>
<tr>
<td>power on page 239</td>
<td>Mathematical</td>
<td>The value that results from raising the specified number to a given power.</td>
</tr>
<tr>
<td>proc_role on page 240</td>
<td>Security</td>
<td>1 if the user has the correct role to execute the procedure; 0 if the user does not have this role.</td>
</tr>
<tr>
<td>pssinfo on page 241</td>
<td>System</td>
<td>Returns information from the Adaptive Server process status structure (pss)</td>
</tr>
<tr>
<td>radians on page 243</td>
<td>Mathematical</td>
<td>The size, in radians, of an angle with a specified number of degrees.</td>
</tr>
<tr>
<td>rand on page 244</td>
<td>Mathematical</td>
<td>A random value between 0 and 1, generated using the specified seed value.</td>
</tr>
<tr>
<td>replicate on page 246</td>
<td>String</td>
<td>A string consisting of the specified expression repeated a given number of times.</td>
</tr>
<tr>
<td>reserved_pages on page 250</td>
<td>System</td>
<td>The number of pages allocated to the specified table or index.</td>
</tr>
<tr>
<td>reverse on page 254</td>
<td>String</td>
<td>The specified string, with characters listed in reverse order.</td>
</tr>
<tr>
<td>right on page 255</td>
<td>String</td>
<td>The part of the character expression, starting the specified number of characters from the right.</td>
</tr>
<tr>
<td>rm_appcontext on page 257</td>
<td>Security</td>
<td>Removes a specific application context, or all application contexts.</td>
</tr>
<tr>
<td>Function</td>
<td>Type</td>
<td>Return value</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>role_contain on page 258</td>
<td>Security</td>
<td>1 if role2 contains role1.</td>
</tr>
<tr>
<td>role_id on page 259</td>
<td>Security</td>
<td>The system role ID of the role with the name you specify.</td>
</tr>
<tr>
<td>role_name on page 260</td>
<td>Security</td>
<td>The name of a role with the system role ID you specify.</td>
</tr>
<tr>
<td>round on page 261</td>
<td>Mathematical</td>
<td>The value of the specified number, rounded to a given number of decimal places.</td>
</tr>
<tr>
<td>row_count on page 263</td>
<td>System</td>
<td>An estimate of the number of rows in the specified table.</td>
</tr>
<tr>
<td>rtrim on page 264</td>
<td>String</td>
<td>The specified expression, trimmed of trailing blanks.</td>
</tr>
<tr>
<td>set_appcontext on page 266</td>
<td>Security</td>
<td>Sets an application context name, attribute name, and attribute value for a user session, defined by the attributes of a specified application.</td>
</tr>
<tr>
<td>show_role on page 268</td>
<td>Security</td>
<td>The login's currently active roles.</td>
</tr>
<tr>
<td>show_sec_services on page 269</td>
<td>Security</td>
<td>A list of the user's currently active security services.</td>
</tr>
<tr>
<td>sign on page 270</td>
<td>Mathematical</td>
<td>The sign (+1 for positive, 0, or -1 for negative) of the specified value.</td>
</tr>
<tr>
<td>sin on page 271</td>
<td>Mathematical</td>
<td>The sine of the specified angle (in radians).</td>
</tr>
<tr>
<td>sortkey on page 272</td>
<td>System</td>
<td>Values that can be used to order results based on collation behavior, which allows you to work with character collation behaviors beyond the default set of Latin-character-based dictionary sort orders and case or accent sensitivity.</td>
</tr>
<tr>
<td>soundex on page 277</td>
<td>String</td>
<td>A 4-character code representing the way an expression sounds.</td>
</tr>
<tr>
<td>space on page 278</td>
<td>String</td>
<td>A string consisting of the specified number of single-byte spaces.</td>
</tr>
<tr>
<td>square on page 280</td>
<td>Mathematical</td>
<td>Returns the square of a specified value expressed as a float.</td>
</tr>
<tr>
<td>sqrt on page 281</td>
<td>Mathematical</td>
<td>The square root of the specified number.</td>
</tr>
<tr>
<td>str on page 289</td>
<td>String</td>
<td>The character equivalent of the specified number.</td>
</tr>
<tr>
<td>str_replace on page 291</td>
<td>String</td>
<td>Replaces any instances of the second string expression that occur within the first string expression with a third expression.</td>
</tr>
<tr>
<td>stuff on page 295</td>
<td>String</td>
<td>The string formed by deleting a specified number of characters from one string and replacing them with another string.</td>
</tr>
<tr>
<td>substring on page 297</td>
<td>String</td>
<td>The string formed by extracting a specified number of characters from another string.</td>
</tr>
<tr>
<td>sum on page 299</td>
<td>Aggregate</td>
<td>The total of the values.</td>
</tr>
<tr>
<td>suser_id on page 302</td>
<td>System</td>
<td>The server user's ID number from the syslogins system table.</td>
</tr>
<tr>
<td>suser_name on page 302</td>
<td>System</td>
<td>The name of the current server user, or the user where the server user ID is specified.</td>
</tr>
<tr>
<td>syb_quit on page 303</td>
<td>System</td>
<td>Terminates the connection.</td>
</tr>
<tr>
<td>syb_sendmsg on page 304</td>
<td>System</td>
<td>Sends a message to a User Datagram Protocol (UDP) port.</td>
</tr>
<tr>
<td>tan on page 306</td>
<td>Mathematical</td>
<td>The tangent of the specified angle (in radians).</td>
</tr>
</tbody>
</table>
The following sections describe the types of functions in detail. The remainder of the chapter contains descriptions of the individual functions in alphabetical order.
Aggregate functions

The aggregate functions generate summary values that appear as new columns in the query results. The aggregate functions are:

- `avg`
- `count`
- `count_big`
- `max`
- `min`
- `sum`

Aggregate functions can be used in the select list or the having clause of a select statement or subquery. They cannot be used in a where clause.

Each aggregate in a query requires its own worktable. Therefore, a query using aggregates cannot exceed the maximum number of worktables allowed in a query (46).

When an aggregate function is applied to a char datatype value, it implicitly converts the value to varchar, stripping all trailing blanks. Likewise, a unichar datatype value is implicitly converted to univarchar.

The `max`, `min`, and `count` aggregate functions have semantics that include the unichar datatype.

Aggregates used with `group by`

Aggregates are often used with `group by`. With `group by`, the table is divided into groups. Aggregates produce a single value for each group. Without `group by`, an aggregate function in the select list produces a single value as a result, whether it is operating on all the rows in a table or on a subset of rows defined by a where clause.
Aggregate functions and NULL values

Aggregate functions calculate the summary values of the non-null values in a particular column. If the ansinull option is set off (the default), there is no warning when an aggregate function encounters a null. If ansinull is set on, a query returns the following SQLSTATE warning when an aggregate function encounters a null:

**Warning- null value eliminated in set function**

Vector and scalar aggregates

Aggregate functions can be applied to all the rows in a table, in which case they produce a single value, a scalar aggregate. They can also be applied to all the rows that have the same value in a specified column or expression (using the group by and, optionally, the having clause), in which case, they produce a value for each group, a vector aggregate. The results of the aggregate functions are shown as new columns.

You can nest a vector aggregate inside a scalar aggregate. For example:

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

<table>
<thead>
<tr>
<th>type</th>
<th>avg(price)</th>
<th>avg(avg(price))</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNDECIDED</td>
<td>NULL</td>
<td>15.23</td>
</tr>
<tr>
<td>business</td>
<td>13.73</td>
<td>15.23</td>
</tr>
<tr>
<td>mod_cook</td>
<td>11.49</td>
<td>15.23</td>
</tr>
<tr>
<td>popular_comp</td>
<td>21.48</td>
<td>15.23</td>
</tr>
<tr>
<td>psychology</td>
<td>13.50</td>
<td>15.23</td>
</tr>
<tr>
<td>trad_cook</td>
<td>15.96</td>
<td>15.23</td>
</tr>
</tbody>
</table>

(6 rows affected)

The group by clause applies to the vector aggregate—in this case, avg(price). The scalar aggregate, avg(avg(price)), is the average of the average prices by type in the titles table.

In standard SQL, when a `select_list` includes an aggregate, all the `select_list` columns must either have aggregate functions applied to them or be in the `group by` list. Transact-SQL has no such restrictions.
Aggregate functions

Example 1 shows a select statement with the standard restrictions. Example 2 shows the same statement with another item (title_id) added to the select list. order by is also added to illustrate the difference in displays. These “extra” columns can also be referenced in a having clause.

Example 1

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

<table>
<thead>
<tr>
<th>type</th>
<th>avg(price)</th>
<th>avg(advance)</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNDECIDED</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>business</td>
<td>13.73</td>
<td>6,281.25</td>
</tr>
<tr>
<td>mod_cook</td>
<td>11.49</td>
<td>7,500.00</td>
</tr>
<tr>
<td>popular_comp</td>
<td>21.48</td>
<td>7,500.00</td>
</tr>
<tr>
<td>psychology</td>
<td>13.50</td>
<td>4,255.00</td>
</tr>
<tr>
<td>trad_cook</td>
<td>15.96</td>
<td>6,333.33</td>
</tr>
</tbody>
</table>

(6 rows affected)

Example 2

You can use either a column name or any other expression (except a column heading or alias) after group by.

Null values in the group by column are placed into a single group.

```sql
select type, title_id, avg(price), avg(advance)
from titles
order by type
```

<table>
<thead>
<tr>
<th>type</th>
<th>title_id</th>
<th>avg(price)</th>
<th>avg(advance)</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNDECIDED</td>
<td>MC3026</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>business</td>
<td>BU1032</td>
<td>13.73</td>
<td>6,281.25</td>
</tr>
<tr>
<td>business</td>
<td>BU1111</td>
<td>13.73</td>
<td>6,281.25</td>
</tr>
<tr>
<td>business</td>
<td>BU2075</td>
<td>13.73</td>
<td>6,281.25</td>
</tr>
<tr>
<td>business</td>
<td>BU7832</td>
<td>13.73</td>
<td>6,281.25</td>
</tr>
<tr>
<td>mod_cook</td>
<td>MC2222</td>
<td>11.49</td>
<td>7,500.00</td>
</tr>
<tr>
<td>mod_cook</td>
<td>MC3021</td>
<td>11.49</td>
<td>7,500.00</td>
</tr>
<tr>
<td>popular_comp</td>
<td>PC1035</td>
<td>21.48</td>
<td>7,500.00</td>
</tr>
<tr>
<td>popular_comp</td>
<td>PC8888</td>
<td>21.48</td>
<td>7,500.00</td>
</tr>
<tr>
<td>popular_comp</td>
<td>PC9999</td>
<td>21.48</td>
<td>7,500.00</td>
</tr>
<tr>
<td>psychology</td>
<td>PS1372</td>
<td>13.50</td>
<td>4,255.00</td>
</tr>
<tr>
<td>psychology</td>
<td>PS2091</td>
<td>13.50</td>
<td>4,255.00</td>
</tr>
<tr>
<td>psychology</td>
<td>PS2106</td>
<td>13.50</td>
<td>4,255.00</td>
</tr>
<tr>
<td>psychology</td>
<td>PS3333</td>
<td>13.50</td>
<td>4,255.00</td>
</tr>
<tr>
<td>psychology</td>
<td>PS7777</td>
<td>13.50</td>
<td>4,255.00</td>
</tr>
<tr>
<td>trad_cook</td>
<td>TC3218</td>
<td>15.96</td>
<td>6,333.33</td>
</tr>
</tbody>
</table>
Example 3

The compute clause in a select statement uses row aggregates to produce summary values. The row aggregates make it possible to retrieve detail and summary rows with one command. Example 3 illustrates this feature:

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

<table>
<thead>
<tr>
<th>type</th>
<th>title_id</th>
<th>price</th>
<th>advance</th>
</tr>
</thead>
<tbody>
<tr>
<td>psychology</td>
<td>PS1372</td>
<td>21.59</td>
<td>7,000.00</td>
</tr>
<tr>
<td>psychology</td>
<td>PS2091</td>
<td>19.99</td>
<td>2,000.00</td>
</tr>
<tr>
<td>psychology</td>
<td>PS2106</td>
<td>7.00</td>
<td>6,000.00</td>
</tr>
<tr>
<td>psychology</td>
<td>PS3333</td>
<td>19.99</td>
<td>2,000.00</td>
</tr>
<tr>
<td>psychology</td>
<td>PS7777</td>
<td>7.99</td>
<td>4,000.00</td>
</tr>
<tr>
<td>sum</td>
<td></td>
<td>67.52</td>
<td>21,275.00</td>
</tr>
</tbody>
</table>

Note the difference in display between Example 3 and the examples without compute (Example 1 and Example 2).

You cannot use aggregate functions on virtual tables such as sysprocesses and syslocks.

If you include an aggregate function in the select clause of a cursor, that cursor cannot be updated.

### Aggregate functions as row aggregates

Row aggregate functions generate summary values that appear as additional rows in the query results.

To use the aggregate functions as row aggregates, use the following syntax:

```sql
Start of select statement
compute row_aggregate(column_name)
[... . row_aggregate(column_name)]...
[by column_name [ , column_name]...]
```

Where:
**Aggregate functions**

- `column_name` – is the name of a column. It must be enclosed in parentheses. Only exact numeric, approximate numeric, and money columns can be used with sum and avg.

  One compute clause can apply the same function to several columns. When using more than one function, use more than one compute clause.

- `by` – indicates that row aggregate values are to be calculated for subgroups. Whenever the value of the by item changes, row aggregate values are generated. If you use by, you must use order by.

  Listing more than one item after by breaks a group into subgroups and applies a function at each level of grouping.

The row aggregates make it possible to retrieve detail and summary rows with one command. The aggregate functions, on the other hand, ordinarily produce a single value for all the selected rows in the table or for each group, and these summary values are shown as new columns.

The following examples illustrate the differences:

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

<table>
<thead>
<tr>
<th>type</th>
<th>sum(price)</th>
<th>sum(advance)</th>
</tr>
</thead>
<tbody>
<tr>
<td>mod_cook</td>
<td>22.98</td>
<td>15,000.00</td>
</tr>
<tr>
<td>trad_cook</td>
<td>47.89</td>
<td>19,000.00</td>
</tr>
</tbody>
</table>

(2 rows affected)
```

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

<table>
<thead>
<tr>
<th>type</th>
<th>price</th>
<th>advance</th>
</tr>
</thead>
<tbody>
<tr>
<td>mod_cook</td>
<td>2.99</td>
<td>15,000.00</td>
</tr>
<tr>
<td>mod_cook</td>
<td>19.99</td>
<td>0.00</td>
</tr>
<tr>
<td>sum</td>
<td>sum</td>
<td></td>
</tr>
<tr>
<td></td>
<td>22.98</td>
<td>15,000.00</td>
</tr>
<tr>
<td>trad_cook</td>
<td>11.95</td>
<td>4,000.00</td>
</tr>
</tbody>
</table>
```
The columns in the compute clause must appear in the select list.

The order of columns in the select list overrides the order of the aggregates in the compute clause. For example:

```sql
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
```

(7 rows affected)
Statistical aggregate functions

If the ansinull option is set off (the default), there is no warning when a row aggregate encounters a null. If ansinull is set on, a query returns the following SQLSTATE warning when a row aggregate encounters a null:

```
Warning - null value eliminated in set function
```

You cannot use select into in the same statement as a compute clause because there is no way to store the compute clause output in the resulting table.

Statistical aggregate functions

Aggregate functions summarize data over a group of rows from the database. The groups are formed using the group by clause of the select statement.

Simple aggregate functions, such as sum, avg, max, min, count_big, and count are allowed only in the select list and in the having and order by clauses as well as the compute clause of a select statement. These functions summarize data over a group of rows from the database.

Adaptive Server Enterprise supports statistical aggregate functions, which permit statistical analysis of numeric data. These functions include stddev, stddev_samp, stddev_pop, variance, var_samp, and var_pop.

These functions, including stddev and variance, are true aggregate functions in that they can compute values for a group of rows as determined by the query’s group by clause. As with other basic aggregate functions such as max or min, their computation ignores null values in the input. Also, regardless of the domain of the expression being analyzed, all variance and standard deviation computation uses IEEE double-precision floating-point standard.

If the input to any variance or standard deviation function is the empty set, then each function returns as its result a null value. If the input to any variance or standard deviation function is a single value, then each function returns 0 as its result.

Standard deviation and variance

The statistical aggregate functions (and their aliases) are:
• *stddev_pop* (also *stddevp*) – standard deviation of a population. Computes the population standard deviation of the provided value expression evaluated for each row of the group (if distinct was specified, then each row that remains after duplicates have been eliminated), defined as the square root of the population variance. See *stddev_pop* on page 285 for syntax and usage information.

• *stddev_samp* (also *stdev*, *stddev*) – standard deviation of a sample. Computes the population standard deviation of the provided value expression evaluated for each row of the group (if distinct was specified, then each row that remains after duplicates have been eliminated), defined as the square root of the sample variance. See *stddev_samp* on page 287 for syntax and usage information.

• *var_pop* (also *varp*) – variance of a population. Computes the population variance of value expression evaluated for each row of the group (if distinct was specified, then each row that remains after duplicates have been eliminated), defined as the sum of squares of the difference of value expression from the mean of value expression, divided by the number of rows in the group. See *var_pop* on page 327 for syntax and usage information.

• *var_samp* (also *var*, *variance*) – variance of a sample. Computes the sample variance of value expression evaluated for each row of the group (if distinct was specified, then each row that remains after duplicates have been eliminated), defined as the sum of squares of the difference from the mean of the value expression, divided by one less than the number of rows in the group. See *var_samp* on page 329 for syntax and usage information.

**Statistical aggregates**

Statistical aggregates are similar to the *avg* aggregate in that:

• The syntax is:

  ```
  var_pop [(all | distinct) expression]
  ```

• Only expressions with numerical datatypes are valid.

• Null values do not participate in the calculation.

• The result is NULL only if no data participates in the calculation.

• The distinct or all clauses can precede the expression (the default is all).
Statistical aggregate functions

- You can use statistical aggregates as vector aggregates (with group by), scalar aggregates (without group by), or in the compute clause.

Unlike the avg aggregate, however, the results are:

- Always of float datatype (that is, a double-precision floating-point), whereas for the avg aggregate, the datatype of the result is the same as that of the expression (with exceptions).
- 0.0 for a single data point.

Formulas

**Figure 2-1: The formula for population-related statistical aggregate functions**

The formula that defines the variance of the population of size \( n \) having mean \( \mu \) (var_pop) is as follows. The population standard deviation (stddev_pop) is the positive square root of this.

\[
\sigma^2 = \frac{\sum (x_i - \mu)^2}{n} \quad \sigma^2 = \text{Variance} \\
n = \text{Population size} \\
\mu = \text{Mean of the values } x_i
\]

**Figure 2-2: The formula for sample-related statistical aggregate functions**

The formula that defines an unbiased estimate of the population variance from a sample of size \( n \) having mean \( \bar{x} \) (var_samp) is as follows. The sample standard deviation (stddev_samp) is the positive square root of this.

\[
s^2 = \frac{\sum (x_i - \bar{x})^2}{n - 1} \quad s^2 = \text{Variance} \\
n = \text{Sample size} \\
\bar{x} = \text{Mean of the values } x_i
\]

The essential difference between the two formulas is the division by \( n-1 \) instead of \( n \).

These two functions are similar, but are used for different purposes:

- var_samp – is used when you want evaluate a sample—that is, a subset—of a population as being representative of the entire population
- var_pop – is used when you have all of the data available for a population, or when \( n \) is so large that the difference between \( n \) and \( n-1 \) is insignificant
Datatype conversion functions

Datatype conversion functions change expressions from one datatype to another and specify new display formats for date and time information. The datatype conversion functions are:

- `cast`
- `convert`
- `inttohex`
- `hextoint`
- `hextobigint`
- `biginttohex`
- `str`

You can use the datatype conversion functions in the select list, in the `where` clause, and anywhere else an expression is allowed.

Adaptive Server performs certain datatype conversions automatically. These are called **implicit conversions**. For example, if you compare a `char` expression and a `datetime` expression, or a `smallint` expression and an `int` expression, or `char` expressions of different lengths, Adaptive Server automatically converts one datatype to another.

You must request other datatype conversions explicitly, using one of the built-in datatype conversion functions. For example, before concatenating numeric expressions, you must convert them to character expressions.

Adaptive Server does not allow you to convert certain datatypes to certain other datatypes, either implicitly or explicitly. For example, you cannot convert the following:

- `smallint` data to `datetime`
- `datetime` data to `smallint`
- `binary` or `varbinary` data to `smalldatetime` or `datetime` data

Unsupported conversions result in error messages.

**bigtime and bigdatetime datatype conversions**

Implicit conversion are allowed between new and existing chronological datatypes.
Implicit conversion between types whose primary fields do not match may cause either data truncation, the insertion of a default value, or an error message to be raised. For example, when a `bigdatetime` value is converted to a `date` value, the time portion is truncated leaving only the date portion. If a `bigtime` value is converted to a `bigdatetime` value, a default date portion of Jan 1, 0001 is added to the new `bigdatetime` value. If a `date` value is converted to a `bigdatetime` value, a default time portion of 00:00:00.000000 is added to the `bigdatetime` value.

Implicit and explicit conversions are allowed where a decreased precision results in the loss of data.

Table 2-3 and Table 2-4 indicate whether individual datatype conversions are performed implicitly, explicitly, or are not supported.
Table 2-3: Explicit, implicit, and unsupported datatype conversions for binary – unsigned int

<table>
<thead>
<tr>
<th>From</th>
<th>binary</th>
<th>varbinary</th>
<th>bit</th>
<th>[n]char</th>
<th>[n]varchar</th>
<th>datetime</th>
<th>smalldatetime</th>
<th>bigdatetime</th>
<th>bigint</th>
<th>tinyint</th>
<th>smallint</th>
<th>unsigned smallint</th>
<th>int</th>
<th>unsigned int</th>
</tr>
</thead>
<tbody>
<tr>
<td>binary</td>
<td>-</td>
<td>I I I I I I I I I I I I I I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>varbinary</td>
<td>I</td>
<td>- I I I I I I I I I I I I I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bit</td>
<td>I I</td>
<td>- I I U U U U I I I I I I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[n]char</td>
<td>I I E -</td>
<td>I I I I I E E E E E E E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[n]varchar</td>
<td>I I E I -</td>
<td>I I I I E E E E E E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>datetime</td>
<td>I I U I I I I - I I I U U U U U</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>smalldatetime</td>
<td>I I U I I I I -</td>
<td>I I I U U U U U</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bigdatetime</td>
<td>I I U I I I I I I I I I I I -</td>
<td>U U U U U U U</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bigint</td>
<td>I I U I I I I I I I - I I I I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tinyint</td>
<td>I I I E E U U U U - I I I I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>smallint</td>
<td>I I I E E U U U U U I - I I I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>unsigned smallint</td>
<td>I I I E E U U U U U I I I I I -</td>
<td>I I I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>int</td>
<td>I I I E E U U U U U I I I I I -</td>
<td>I I I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>unsigned int</td>
<td>I I I E E U U U U U I I I I I -</td>
<td>I I I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bigint</td>
<td>I I I E E U U U U U I I I I I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>unsigned bigint</td>
<td>I I I E E U U U U U I I I I I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decimal</td>
<td>I I I E E U U U U U I I I I I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>numeric</td>
<td>I I I E E U U U U U I I I I I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>float</td>
<td>I I I E E U U U U U I I I I I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>real</td>
<td>I I I E E U U U U U I I I I I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>money</td>
<td>I I I I I U U U U U I I I I I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>smallmoney</td>
<td>I I I I I U U U U U I I I I I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>text</td>
<td>U U U E E U U U U U U U U U</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>unitext</td>
<td>E E E E E E U U U U U U U U</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>image</td>
<td>E E U U U U U U U U U U U U</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>unichar</td>
<td>I I E I I I I I I I E E E E E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>univarchar</td>
<td>I I E I I I I I I I E E E E E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>date</td>
<td>I I U I I I U I U U U U U U</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>I I U I I I U I U U U U U U</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 2-4: Explicit, implicit, and unsupported datatype conversions for bigint – time

<table>
<thead>
<tr>
<th>From</th>
<th>bigint</th>
<th>unsigned bigint</th>
<th>decimal</th>
<th>numeric</th>
<th>float</th>
<th>real</th>
<th>money</th>
<th>smallmoney</th>
<th>text</th>
<th>unibyte</th>
<th>image</th>
<th>unichar</th>
<th>univarchar</th>
<th>date</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>binary</td>
<td>I I I I I I I I I I I I</td>
<td>I I I I I I I I I I I I</td>
<td>I I I</td>
<td>I I I</td>
<td>I I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
</tr>
<tr>
<td>varbinary</td>
<td>I I I I I I I I I I I I</td>
<td>I I I I I I I I I I I I</td>
<td>I I I</td>
<td>I I I</td>
<td>I I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
</tr>
<tr>
<td>bit</td>
<td>I I I I I I I I I I U U U E E U</td>
<td>I I I</td>
<td>I I I</td>
<td>I I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
</tr>
<tr>
<td>[n]char</td>
<td>E E E E E E E E E E</td>
<td>I I I</td>
<td>I I I</td>
<td>I I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
</tr>
<tr>
<td>[n]varchar</td>
<td>E E E E E E E E E E</td>
<td>I I I</td>
<td>I I I</td>
<td>I I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
</tr>
<tr>
<td>datetime</td>
<td>U U U U U U U U U U U U I I I I</td>
<td>I I I</td>
<td>I I I</td>
<td>I I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
</tr>
<tr>
<td>smalldatetime</td>
<td>U U U U U U U U U U U U U U U</td>
<td>I I I</td>
<td>I I I</td>
<td>I I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
</tr>
<tr>
<td>bigdatetime</td>
<td>U U U U U U U U U U U U U U U</td>
<td>I I I</td>
<td>I I I</td>
<td>I I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
</tr>
<tr>
<td>bigtime</td>
<td>U U U U U U U U U U U U U U U</td>
<td>I I I</td>
<td>I I I</td>
<td>I I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
</tr>
<tr>
<td>tinyint</td>
<td>I I I I I I I I I I I</td>
<td>I I I</td>
<td>I I I</td>
<td>I I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
</tr>
<tr>
<td>smallint</td>
<td>I I I I I I I I I I U U U E E U</td>
<td>I I I</td>
<td>I I I</td>
<td>I I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
</tr>
<tr>
<td>unsigned smallint</td>
<td>I I I I I I I I I I U U U E E U</td>
<td>I I I</td>
<td>I I I</td>
<td>I I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
</tr>
<tr>
<td>int</td>
<td>I I I I I I I I I I U U U E E U</td>
<td>I I I</td>
<td>I I I</td>
<td>I I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
</tr>
<tr>
<td>unsigned int</td>
<td>I I I I I I I I I I U U U E E U</td>
<td>I I I</td>
<td>I I I</td>
<td>I I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
</tr>
<tr>
<td>bigint</td>
<td>– I I I I I I I I I I U U U E E U</td>
<td>I I I</td>
<td>I I I</td>
<td>I I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
</tr>
<tr>
<td>unsigned bigint</td>
<td>– I I I I I I I I I I U U U E E U</td>
<td>I I I</td>
<td>I I I</td>
<td>I I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
</tr>
<tr>
<td>decimal</td>
<td>I I I I I I I I I I U U U E E U</td>
<td>I I I</td>
<td>I I I</td>
<td>I I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
</tr>
<tr>
<td>numeric</td>
<td>I I I I I I I I I I U U U E E U</td>
<td>I I I</td>
<td>I I I</td>
<td>I I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
</tr>
<tr>
<td>float</td>
<td>I I I I I I I I I I U U U E E U</td>
<td>I I I</td>
<td>I I I</td>
<td>I I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
</tr>
<tr>
<td>real</td>
<td>I I I I I I I I I I U U U E E U</td>
<td>I I I</td>
<td>I I I</td>
<td>I I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
</tr>
<tr>
<td>money</td>
<td>I I I I I I I I I I I I I</td>
<td>I I I</td>
<td>I I I</td>
<td>I I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
</tr>
<tr>
<td>smallmoney</td>
<td>I I I I I I I I I I I I</td>
<td>I I I</td>
<td>I I I</td>
<td>I I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
</tr>
<tr>
<td>text</td>
<td>U U U U U U U U U U U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
</tr>
<tr>
<td>unibyte</td>
<td>U U U U U U U U U U U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
</tr>
<tr>
<td>image</td>
<td>U U U U U U U U U U U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
</tr>
<tr>
<td>unichar</td>
<td>E E E E E E E E E E E E E</td>
<td>E E E E E E E E E E E E E</td>
<td>I I I</td>
<td>I I I</td>
<td>I I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
</tr>
<tr>
<td>univarchar</td>
<td>E E E E E E E E E E E E E</td>
<td>E E E E E E E E E E E E E</td>
<td>I I I</td>
<td>I I I</td>
<td>I I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
<td>I I</td>
</tr>
<tr>
<td>date</td>
<td>U U U U U U U U U U U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
</tr>
<tr>
<td>time</td>
<td>U U U U U U U U U U U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
<td>U U</td>
</tr>
</tbody>
</table>
key

- E – explicit datatype conversion is required.
- I – conversion can be done either implicitly, or with an explicit datatype conversion function.
- I/E – Explicit datatype conversion function required when there is loss of precision or scale, and arithabortonumeric_truncation is on; implicit conversion allowed otherwise.
- U – unsupported conversion.
- – conversion of a datatype to itself. These conversions are allowed, but are meaningless.

Converting character data to a noncharacter type

You can convert character data to a noncharacter type—such as a money, date/time, exact numeric, or approximate numeric type—if it consists entirely of characters that are valid for the new type. Leading blanks are ignored. However, if a char expression that consists of a blank or blanks is converted to a datetime expression, Adaptive Server converts the blanks into the default datetime value of “Jan 1, 1900.”

Syntax errors are generated when the data includes unacceptable characters. Following are some examples of characters that cause syntax errors:

- Commas or decimal points in integer data
- Commas in monetary data
- Letters in exact or approximate numeric data or bit stream data
- Misspelled month names in date and time data

Implicit conversions between unichar/univarchar and datetime/smalldatetime are supported.

Converting from one character type to another

When converting from a multibyte character set to a single-byte character set, characters with no single-byte equivalent are converted to question marks.
text and unitext columns can be explicitly converted to char, nchar, varchar, unichar, univarchar, or nvarchar. You are limited to the maximum length of the character datatypes, which is determined by the maximum column size for your server’s logical page size. If you do not specify the length, the converted value has a default length of 30 bytes.

Converting numbers to a character type

Exact and approximate numeric data can be converted to a character type. If the new type is too short to accommodate the entire string, an insufficient space error is generated. For example, the following conversion tries to store a 5-character string in a 1-character type:

```sql
select convert(char(1), 12.34)
```

Insufficient result space for explicit conversion of NUMERIC value '12.34' to a CHAR field.

When converting float data to a character type, the new type should be at least 25 characters long.

**Note** The `str` function may be preferable to `convert` or `cast` when making conversions, because it provides more control over conversions and avoids errors.

Rounding during conversion to and from money types

The money and smallmoney types store 4 digits to the right of the decimal point, but round up to the nearest hundredth (.01) for display purposes. When data is converted to a money type, it is rounded up to four places.

Data converted from a money type follows the same rounding behavior if possible. If the new type is an exact numeric with less than three decimal places, the data is rounded to the scale of the new type. For example, when $4.50 is converted to an integer, it yields 5:

```sql
select convert(int, $4.50)
```

-----------

       5
Data converted to money or smallmoney is assumed to be in full currency units such as dollars rather than in fractional units such as cents. For example, the integer value of 5 is converted to the money equivalent of 5 dollars, not 5 cents, in the us_english language.

**Converting date and time information**

Data that is recognizable as a date can be converted to datetime, smalldatetime, date, or time. Incorrect month names lead to syntax errors. Dates that fall outside the acceptable range for the datatype lead to arithmetic overflow errors.

When datetime values are converted to smalldatetime, they are rounded to the nearest minute.

When converting date data to a character type, use style numbers 1 through 7 (101 through 107) or 10 through 12 (110 through 112) in Table 2-7 on page 119 to specify the display format. The default value is 100 (mon dd yyyy hh:miAM (or PM)). If date data is converted to a style that contains a time portion, that time portion reflects the default value of zero.

When converting time data to a character type, use style number 8 or 9 (108 or 109) to specify the display format. The default is 100 (mon dd yyyy hh:miAM (or PM)). If time data is converted to a style that contains a date portion, the default date of Jan 1, 1900 is displayed.

**Converting between numeric types**

You can convert data from one numeric type to another. Errors can occur if the new type is an exact numeric with precision or scale that is not sufficient to hold the data.

For example, if you provide a float or numeric value as an argument to a built-in function that expects an integer, the value of the float or numeric is truncated. However, Adaptive Server does not implicitly convert numerics that have a fractional part but returns a scale error message. For example, Adaptive Server returns error 241 for numerics that have a fractional part and error 257 if other datatypes are passed.

Use the arithabort and arithignore options to determine how Adaptive Server handles errors resulting from numeric conversions.
Datatype conversion functions

Arithmetic overflow and divide-by-zero errors

Divide-by-zero errors occur when Adaptive Server tries to divide a numeric value by zero. Arithmetic overflow errors occur when the new type has too few decimal places to accommodate the results. This happens during:

- Explicit or implicit conversions to exact types with a lower precision or scale
- Explicit or implicit conversions of data that falls outside the acceptable range for a money or date/time type
- Conversions of hexadecimal strings requiring more than 4 bytes of storage using hextoint

Both arithmetic overflow and divide-by-zero errors are considered serious, whether they occur during an implicit or explicit conversion. Use the arithabort arith_overflow option to determine how Adaptive Server handles these errors. The default setting, arithabort arith_overflow on, rolls back the entire transaction in which the error occurs. If the error occurs in a batch that does not contain a transaction, arithabort arith_overflow on does not roll back earlier commands in the batch, and Adaptive Server does not execute statements that follow the error-generating statement in the batch. If you set arithabort arith_overflow off, Adaptive Server aborts the statement that causes the error, but continues to process other statements in the transaction or batch. You can use the @@error global variable to check statement results.

Use the arithignore arith_overflow option to determine whether Adaptive Server displays a message after these errors. The default setting, off, displays a warning message when a divide-by-zero error or a loss of precision occurs. Setting arithignore arith_overflow on suppresses warning messages after these errors. You can omit optional arith_overflow keyword without any effect.

Scale errors

When an explicit conversion results in a loss of scale, the results are truncated without warning. For example, when you explicitly convert a float, numeric, or decimal type to an integer, Adaptive Server assumes you want the result to be an integer and truncates all numbers to the right of the decimal point.
During implicit conversions to numeric or decimal types, loss of scale generates a scale error. Use the arithabortion numeric_truncation option to determine how serious such an error is considered. The default setting, arithabortion numeric_truncation on, aborts the statement that causes the error, but continues to process other statements in the transaction or batch. If you set arithabortion numeric_truncation off, Adaptive Server truncates the query results and continues processing.

**Note** For entry level ANSI SQL compliance, set:

- `arithabortion arith_overflow off`
- `arithabortion numeric_truncation on`
- `arithignore off`

### Domain errors

The `convert` function generates a domain error when the function’s argument falls outside the range over which the function is defined. This happens rarely.

### Conversions between binary and integer types

The `binary` and `varbinary` types store hexadecimal-like data consisting of a “0x” prefix followed by a string of digits and letters.

These strings are interpreted differently by different platforms. For example, the string “0x0000100” represents 65536 on machines that consider byte 0 most significant (little-endian) and 256 on machines that consider byte 0 least significant (big-endian).

Binary types can be converted to integer types either explicitly, using the `convert` function, or implicitly. If the data is too short for the new type, it is stripped of its “0x” prefix and zero-padded. If it is too long, it is truncated.

Both `convert` and the implicit datatype conversions evaluate binary data differently on different platforms. Because of this, results may vary from one platform to another. Use the `hextoint` function for platform-independent conversion of hexadecimal strings to integers, and the `inttohex` function for platform-independent conversion of integers to hexadecimal values. Use the `hextobigint` function for platform-independent conversion of hexadecimal strings to 64-bit integers, and the `biginttohex` function for platform-independent conversion of 64-bit integers to hexadecimal values.
Datatype conversion functions

Converting between binary and numeric or decimal types

In binary and varbinary data strings, the first two digits after “0x” represent the binary type: “00” represents a positive number and “01” represents a negative number. When you convert a binary or varbinary type to numeric or decimal, be sure to specify the “00” or “01” values after the “0x” digit; otherwise, the conversion will fail.

For example, here is how to convert the following binary data to numeric:

```
select convert(numeric(38, 18), 0x000000000000000006b14bd1e6eea000000000000000000000000000)
```

----------

123.456000

This example converts the same numeric data back to binary:

```
select convert(binary, convert(numeric(38, 18), 123.456))
```

--------------------------------------------------------------

0x000000000000000006b14bd1e6eea000000000000000000000000000

Converting image columns to binary types

You can use the convert function to convert an image column to binary or varbinary. You are limited to the maximum length of the binary datatypes, which is determined by the maximum column size for your server’s logical page size. If you do not specify the length, the converted value has a default length of 30 characters.

Converting other types to bit

Exact and approximate numeric types can be converted to the bit type implicitly. Character types require an explicit convert function.

The expression being converted must consist only of digits, a decimal point, a currency symbol, and a plus or minus sign. The presence of other characters generates syntax errors.

The bit equivalent of 0 is 0. The bit equivalent of any other number is 1.
Converting NULL value

You can use the convert function to change NULL to NOT NULL and NOT NULL to NULL.

Date functions

The date functions manipulate values of the datatypes datetime, bigtime, bigdatetime, smalldatetime, date or time.

You can use date functions in the select list or where clause of a query.

Use the datetime datatype only for dates after January 1, 1753. datatime values must be enclosed in single or double quotes. Use date for dates from January, 1, 0001 to January 1, 9999. date values must be enclosed in single or double quotes. Use char, nchar, varchar, or nvarchar for earlier dates. Adaptive Server recognizes a wide variety of date formats. See “Datatype conversion functions” on page 63 and “Date and time datatypes” on page 20 for more information.

Adaptive Server automatically converts between character and datatime values when necessary (for example, when you compare a character value to a datatime value).

The date datatype can cover dates from January 1, 0001 to January 1, 9999.

Date parts

The date parts, the abbreviations recognized by Adaptive Server, and the acceptable values are:

<table>
<thead>
<tr>
<th>Date part</th>
<th>Abbreviation</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>year</td>
<td>yy</td>
<td>1753 – 9999 (datetime)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1753 – 2079 (smalldatetime)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0001 – 9999 (bigdatetime)</td>
</tr>
<tr>
<td>quarter</td>
<td>qq</td>
<td>1 – 4</td>
</tr>
<tr>
<td>month</td>
<td>mm</td>
<td>1 – 12</td>
</tr>
<tr>
<td>week</td>
<td>wk</td>
<td>1 – 54</td>
</tr>
<tr>
<td>day</td>
<td>dd</td>
<td>1 – 31</td>
</tr>
<tr>
<td>dayofyear</td>
<td>dy</td>
<td>1 – 366</td>
</tr>
</tbody>
</table>
When you enter a year as two digits (yy):

- Numbers less than 50 are interpreted as 20yy. For example, 01 is 2001, 32 is 2032, and 49 is 2049.
- Numbers equal to or greater than 50 are interpreted as 19yy. For example, 50 is 1950, 74 is 1974, and 99 is 1999.

For datetime, smalldatetime and time values milliseconds can be preceded either with a colon or a period. If preceded by a colon, the number means thousandths of a second. If preceded by a period, a single digit means tenths of a second, two digits mean hundredths of a second, and three digits mean thousandths of a second. For example, “12:30:20:1” means twenty and one-thousandth of a second past 12:30; “12:30:20.1” means twenty and one-tenth of a second past 12:30. Adaptive Server may round or truncate millisecond values when inserting datetime or time data, as these datatypes have a granularity of 1/300th of a second rather than 1/1000th of a second. You can use the time datatype for time information.

For bigdatetime and bigtime time values, microseconds must be preceded by a decimal point and represent fractional seconds.

### Mathematical functions

Mathematical functions return values commonly needed for operations on mathematical data. Mathematical function names are not keywords.

Each function also accepts arguments that can be implicitly converted to the specified type. For example, functions that accept approximate numeric types also accept integer types. Adaptive Server automatically converts the argument to the desired type.

The mathematical functions are:

<table>
<thead>
<tr>
<th>Date part</th>
<th>Abbreviation</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>weekday</td>
<td>dw</td>
<td>1 – 7 (Sun. – Sat.)</td>
</tr>
<tr>
<td>hour</td>
<td>hh</td>
<td>0 – 23</td>
</tr>
<tr>
<td>minute</td>
<td>mi</td>
<td>0 – 59</td>
</tr>
<tr>
<td>second</td>
<td>ss</td>
<td>0 – 59</td>
</tr>
<tr>
<td>millisecond</td>
<td>ms</td>
<td>0 – 999</td>
</tr>
<tr>
<td>microseconds</td>
<td>us</td>
<td>0-999999</td>
</tr>
</tbody>
</table>

Mathematical functions

Mathematical functions return values commonly needed for operations on mathematical data. Mathematical function names are not keywords.

Each function also accepts arguments that can be implicitly converted to the specified type. For example, functions that accept approximate numeric types also accept integer types. Adaptive Server automatically converts the argument to the desired type.

The mathematical functions are:
Error traps are provided to handle domain or range errors of these functions. Users can set the arithabort and arithignore options to determine how domain errors are handled:

- `arithabort arith_overflow` specifies behavior following a divide-by-zero error or a loss of precision. The default setting, `arithabort arith_overflow on`, rolls back the entire transaction or aborts the batch in which the error occurs. If you set `arithabort arith_overflow off`, Adaptive Server aborts the statement that causes the error, but continues to process other statements in the transaction or batch.

- `arithabort numeric_truncation` specifies behavior following a loss of scale by an exact numeric type during an implicit datatype conversion. (When an explicit conversion results in a loss of scale, the results are truncated without warning.) The default setting, `arithabort numeric_truncation on`, aborts the statement that causes the error, but continues to process other statements in the transaction or batch. If you set `arithabort numeric_truncation off`, Adaptive Server truncates the query results and continues processing.

- By default, the `arithignore arith_overflow` option is turned off, causing Adaptive Server to display a warning message after any query that results in numeric overflow. Set the `arithignore` option on to ignore overflow errors.

### Security functions

Security functions return security-related information.

The security functions are:
String functions

String function operate on binary data, character strings, and expressions. The string functions are:

- ascii
- char
- charindex
- char_length
- difference
- lower
- ltrim
- patindex
- replicate
- reverse
- right
- rtrim
- soundex
- space
- str
- str_replace
- stuff
- substring
- to_unichar
- uhighsurr
- ulowsurr
- upper
- uscalar

You can nest string functions and use them in a select list, in a where clause, or anywhere an expression is allowed. When you use constants with a string function, enclose them in single or double quotes. String function names are not keywords.

Each string function also accepts arguments that can be implicitly converted to the specified type. For example, functions that accept approximate numeric expressions also accept integer expressions. Adaptive Server automatically converts the argument to the desired type.

When a string function accepts two character expressions but only one expression is unichar, the other expression is “promoted” and internally converted to unichar. This follows existing rules for mixed-mode expressions. However, this conversion may cause truncation, since unichar data sometimes takes twice the space.
CHAPTER 2  Transact-SQL Functions

Limits on string functions

Results of string functions are limited to 16K. This limit is independent of the server’s page size. In Transact-SQL string functions and string variables, literals can be as large as 16K even on a 2K page size.

If set string_rtruncation is on, a user receives an error if an insert or update truncates a character string. However, Adaptive Server does not report an error if a displayed string is truncated. For example:

```
select replicate("a", 16383) + replicate("B", 4000)
```

This shows that the total length would be 20383, but the result string is restricted to 16K.

System functions

System functions return special information from the database. The system functions are:

- `col_length`
- `col_name`
- `curunreservedpgs`
- `data_pages`
- `datalength`
- `db_id`
- `db_name`
- `host_id`
- `host_name`
- `index_col`
- `is_quiesced`
- `isnull`
- `object_id`
- `object_name`
- `reserved_pages`
- `row_count`
- `show_role`
- `suser_id`
- `suser_name`
- `tempdb_id`
- `tran_dumpable_status`
- `tsequal`
- `used_pages`
- `user`
- `user_id`
- `user_name`
- `valid_name`
- `valid_user`

The system functions can be used in a select list, in a where clause, and anywhere an expression is allowed.

When the argument to a system function is optional, the current database, host computer, server user, or database user is assumed.

Text, unitext, and image columns

Text, unitext, and image columns cannot be used:

- As parameters to stored procedures
Text and image functions

- As values passed to stored procedures
- As local variables
- In order by, compute, and group by clauses
- In an index
- In a where clause clause, except with the keyword like
- In joins

In triggers, both the inserted and deleted text values reference the new value; you cannot reference the old value.

Text and image functions

Text and image functions operate on text, image, and unitext data. The text and image functions are:

- `textptr`
- `textvalid`

Text and image built-in function names are not keywords. Use the `set textsize` option to limit the amount of text, image, and unitext data that is retrieved by a `select` statement.

You can use the `patindex` text function on text, image, and unitext columns and can consider it on a text and image function.

You can use the `datalength` function to display the length of data in text, image, and unitext columns.

User-defined SQL functions

You can include these in a scalar function:

- `declare` statements to define data variables and cursors that are local to the function.
- Assigned values to objects local to the function (for example, assigning values to scalar and variables local to a table with `select` or `set` commands).
• Cursor operations that reference local cursors that are declared, opened, closed, and deallocated in the function.

• Control-of-flow statements.

• `set` options (only valid in the scope of the function).

Adaptive Server does not allow `fetch` statements in a scalar function that return data to the client. You cannot include:

• `select` or `fetch` statements that returns data to the client.

• `insert`, `update`, or `delete` statements.

• Utility commands, such as `dbcc`, `dump` and `load` commands.

• `print` statements

• Statement that references `rand`, `rand2`, `getdate`, or `newid`.

You can include `select` or `fetch` statements that assign values only to local variable.
### abs

<table>
<thead>
<tr>
<th>Description</th>
<th>Returns the absolute value of an expression.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td><code>abs(numeric_expression)</code></td>
</tr>
<tr>
<td>Parameters</td>
<td><code>numeric_expression</code> is a column, variable, or expression with datatype that is an exact numeric, approximate numeric, money, or any type that can be implicitly converted to one of these types.</td>
</tr>
<tr>
<td>Examples</td>
<td>Returns the absolute value of -1:</td>
</tr>
<tr>
<td></td>
<td><code>select abs(-1)</code></td>
</tr>
<tr>
<td></td>
<td><code>-----------</code></td>
</tr>
<tr>
<td></td>
<td><code>1</code></td>
</tr>
<tr>
<td>Usage</td>
<td>abs, a mathematical function, returns the absolute value of a given expression. Results are of the same type and have the same precision and scale as the numeric expression.</td>
</tr>
<tr>
<td>Standards</td>
<td>ANSI SQL – Compliance level: Transact-SQL extension.</td>
</tr>
<tr>
<td>Permissions</td>
<td>Any user can execute abs.</td>
</tr>
<tr>
<td>See also</td>
<td>“Mathematical functions” on page 74 for general information about mathematical functions.</td>
</tr>
</tbody>
</table>

**Functions**

- ceiling
- floor
- round
- sign
acos

Description
Returns the angle (in radians) with a specified cosine.

Syntax
acos(cosine)

Parameters
 cosine
is the cosine of the angle, expressed as a column name, variable, or constant
of type float, real, double precision, or any datatype that can be implicitly
converted to one of these types.

Examples
Returns the angle where the cosine is 0.52:

```
select acos(0.52)
--------------------
 1.023945
```

Usage
acos, a mathematical function, returns the angle (in radians) where the cosine
is the specified value.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute acos.

See also
“Mathematical functions” on page 74 for general information about
mathematical functions.

Functions
cos, degrees, radians


**ascii**

**Description**

Returns the ASCII code for the first character in an expression.

**Syntax**

`ascii(char_expr | uchar_expr)`

**Parameters**

`char_expr`

is a character-type column name, variable, or constant expression of `char`, `varchar`, `nchar`, or `nvarchar` type.

`uchar_expr`

is a character-type column name, variable, or constant expression of `unichar` or `univarchar` type.

**Examples**

Returns the author’s last names and the ACSII codes for the first letters in their last names, if the ASCII code is less than 70:

```
select au_lname, ascii(au_lname) from authors
where ascii(au_lname) < 70

au_lname  ascii(au_lname)
-----------  ------------
Bennet      66
Blotchet-Halls  66
Carson      67
DeFrance    68
Dull        68
```

**Usage**

- ascii, a string function, returns the ASCII code for the first character in the expression.

- When a string function accepts two character expressions but only one expression is unichar, the other expression is “promoted” and internally converted to unichar. This follows existing rules for mixed-mode expressions. However, this conversion may cause truncation, since unichar data sometimes takes twice the space.

- If `char_expr` or `uchar_expr` is NULL, returns NULL.

**Standards**

ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**

Any user can execute ascii.

**See also**

For general information about string functions, see “String functions” on page 76.

**Functions**

char, to_unichar
asehostname

Description: Returns the physical or virtual host on which Adaptive Server is running.

Syntax: asehostname

Parameters: None.

Examples: Returns the Adaptive Server host name:

```sql
select asehostname()
-----------------------------------
linuxkernel.sybase.com
```

Standards: SQL/92 and SQL/99 compliant

Permissions: Only users with the sa_role can execute asehostname.
**asin**

**Description**
Returns the angle (in radians) with a specified sine.

**Syntax**
```
asin(sine)
```

**Parameters**
- `sine` is the sine of the angle, expressed as a column name, variable, or constant of type float, real, double precision, or any datatype that can be implicitly converted to one of these types.

**Examples**
```
select asin(0.52)
```

```
-------------------
0.546851
```

**Usage**
asin, a mathematical function, returns the angle (in radians) with a sine of the specified value.

**Standards**
ANSI SQL -- Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute asin.

**See also**
“Mathematical functions” on page 74 for general information about mathematical functions.

**Functions**
degrees, radians, sin
**atan**

<table>
<thead>
<tr>
<th>Description</th>
<th>Returns the angle (in radians) with a specified tangent.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>atan(tangent)</td>
</tr>
<tr>
<td>Parameters</td>
<td>tangent is the tangent of the angle, expressed as a column name, variable, or constant of type float, real, double precision, or any datatype that can be implicitly converted to one of these types.</td>
</tr>
<tr>
<td>Examples</td>
<td>select atan(0.50)</td>
</tr>
<tr>
<td></td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>0.463648</td>
</tr>
</tbody>
</table>

**Usage**
atan, a mathematical function, returns the angle (in radians) of a tangent with the specified value.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute atan.

**See also**
“Mathematical functions” on page 74 for general information about mathematical functions.

**Functions**
atn2, degrees, radians, tan
**atn2**

**Description**
Returns the angle (in radians) with specified sine and cosine.

**Syntax**
```
atn2(sine, cosine)
```

**Parameters**
- **sine**
  is the sine of the angle, expressed as a column name, variable, or constant of type float, real, double precision, or any datatype that can be implicitly converted to one of these types.
- **cosine**
  is the cosine of the angle, expressed as a column name, variable, or constant of type float, real, double precision, or any datatype that can be implicitly converted to one of these types.

**Examples**
```
select atn2(.50, .48)
---------------------
  0.805803
```

**Usage**
atn2, a mathematical function, returns the angle (in radians) whose sine and cosine are specified.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute atn2.

**See also**
“Mathematical functions” on page 74 for general information about mathematical functions.

**Functions**
atan, degrees, radians, tan
**avg**

**Description**
Returns the numeric average of all (distinct) values.

**Syntax**
```
avg([all | distinct] expression)
```

**Parameters**
- **all**
  - applies `avg` to all values. `all` is the default.
- **distinct**
  - eliminates duplicate values before `avg` is applied. `distinct` is optional.

**expression**
- is a column name, constant, function, any combination of column names, constants, and functions connected by arithmetic or bitwise operators, or a subquery. With aggregates, an expression is usually a column name. For more information, see “Expressions” on page 359.

**Examples**

**Example 1** Calculates the average advance and the sum of total sales for all business books. Each of these aggregate functions produces a single summary value for all of the retrieved rows:
```
select avg(advance), sum(total_sales)
from titles
where type = "business"
```

<table>
<thead>
<tr>
<th></th>
<th>6,281.25</th>
<th>30788</th>
</tr>
</thead>
</table>

**Example 2** Used with a `group by` clause, the aggregate functions produce single values for each group, rather than for the entire table. This statement produces summary values for each type of book:
```
select type, avg(advance), sum(total_sales)
from titles
group by type
```

<table>
<thead>
<tr>
<th>type</th>
<th>avg(advance)</th>
<th>sum(total_sales)</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNDECIDED</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>business</td>
<td>6,281.25</td>
<td>30788</td>
</tr>
<tr>
<td>mod_cook</td>
<td>7,500.00</td>
<td>24278</td>
</tr>
<tr>
<td>popular_comp</td>
<td>7,500.00</td>
<td>12875</td>
</tr>
<tr>
<td>psychology</td>
<td>4,255.00</td>
<td>9939</td>
</tr>
<tr>
<td>trad_cook</td>
<td>6,333.33</td>
<td>19566</td>
</tr>
</tbody>
</table>

**Example 3** Groups the titles table by publishers and includes only those groups of publishers who have paid more than $25,000 in total advances and whose books average more than $15 in price:
avg

```
SELECT pub_id, SUM(advance), AVG(price)
FROM titles
GROUP BY pub_id
HAVING SUM(advance) > 25000 AND AVG(price) > 15
```

<table>
<thead>
<tr>
<th>pub_id</th>
<th>advance</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>0877</td>
<td>41,000.00</td>
<td>15.41</td>
</tr>
<tr>
<td>1389</td>
<td>30,000.00</td>
<td>18.98</td>
</tr>
</tbody>
</table>

**Usage**

- `avg`, an aggregate function, finds the average of the values in a column. `avg` can only be used on numeric (integer, floating point, or money) datatypes. Null values are ignored in calculating averages.

- When you average (signed or unsigned) `int`, `smallint`, `tinyint` data, Adaptive Server returns the result as an `int` value. When you average (signed or unsigned) `bigint` data, Adaptive Server returns the result as a `bigint` value. To avoid overflow errors in DB-Library programs, declare variables used for results appropriately.

- You cannot use `avg` with the binary datatypes.

- Since the average value is only defined on numeric datatypes, using `avg` Unicode expressions generates an error.

**Standards**

ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**

Any user can execute `avg`.

**See also**

For general information about aggregate functions, see “Aggregate functions” on page 54.

**Functions**  
`max`, `min`
**audit_event_name**

Description

Returns a description of an audit event.

Syntax

```sql
audit_event_name(event_id)
```

Parameters

- `event_id` is the number of an audit event.

Examples

**Example 1** Queries the audit trail for table creation events:

```sql
select * from audit_data where audit_event_name(event) = "Create Table"
```

**Example 2** Obtains current audit event values. See the Usage section below for a complete list of audit values and their descriptions.

```sql
create table #tmp(event_id int, description varchar(255))
go declare @a int select @a=1 while (@a<120) begin insert #tmp values (@a, audit_event_name(@a)) select @a=@a + 1 end select * from #tmp go
```

<table>
<thead>
<tr>
<th>event_id</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ad hoc Audit Record</td>
</tr>
<tr>
<td>2</td>
<td>Alter Database</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>Create Index</td>
</tr>
<tr>
<td>105</td>
<td>Drop Index</td>
</tr>
</tbody>
</table>

Usage

The following lists the ID and name of each of the audit events:
**audit_event_name**

1. Ad Hoc Audit record
2. Alter Database
3. Alter table
4. BCP In
5. NULL
6. Bind Default
7. Bind Message
8. Bind Rule
9. Create Database
10. Create Table
11. Create Procedure
12. Create Trigger
13. Create Rule
14. Create Default
15. Create Message
16. Create View
17. Access To Database
18. Delete Table
19. Delete View
20. Disk Init
21. Disk Refit
22. Disk Reinit
23. Disk Mirror
24. Disk Unmirror
25. Disk Remirror
26. Drop Database
27. Drop Table
28. Drop Procedure
29. Drop Trigger
30. Drop Rule
31. Drop Default
32. Drop Message
33. Drop View
34. Dump Database
35. Dump Transaction
36. Fatal Error
37. Nonfatal Error
38. Execution Of Stored Procedure
39. Execution Of Trigger
40. Grant Command
41. Insert Table
42. Insert View
43. Load Database
44. Load Transaction
45. Log In
46. Log Out
47. Revoke Command
48. RPC In
49. RPC Out
50. Server Boot
51. Server Shutdown
52. NULL
53. NULL
54. NULL
55. Role Toggling
56. NULL
57. NULL
58. NULL
59. NULL
60. NULL
61. Access To Audit Table
62. Select Table
63. Select View
64. Truncate Table
65. NULL
66. NULL
67. NULL
68. NULL
69. NULL
70. Update Table
71. Update View
72. NULL
73. Auditing Enabled
74. Auditing Disabled
75. NULL
76. SSO Changed Password
77. NULL
78. Config
79. NULL
80. Role Check Performed
81. DBCC Command
82. NULL
83. Online Database
84. NULL
85. User-defined Function Command
86. Built-in Function
87. NULL
88. NULL
89. Reference
90. NULL
91. NULL
92. NULL
93. JCS Install Command
94. JCS Remove Command
95. Unlock Admin Account
96. Quiesce Database Command
97. Create SQLJ Function
98. NULL
99. NULL
100. Disk Resize
101. Mount Database
102. Unmount Database
103. NULL
104. Create Index
105.Drop Index
106. NULL
107. NULL
108. NULL
109. NULL
110. Deploy UDWS
111. Undeploy UDWS
112. NULL
113. NULL
114. NULL
115. Password Administration

**Note** Adaptive Server does not log events if audit_event_name returns NULL.

**Standards**
ANSI SQL – compliance level: Transact-SQL extension.

**Permissions**
Any user can execute audit_event_name.

**See also**
Commands: select, sp_audit
**authmech**

**Description**
Determines what authentication mechanism is used by a specified logged in server process ID.

**Syntax**
authmech (spid)

**Examples**

**Example 1** Returns the authentication mechanism for server process ID 42, whether KERBEROS, LDAP, or any other mechanism:
```
select authmech(42)
```

**Example 2** Returns the authentication mechanism for the current login’s server process ID:
```
select authmech()
```

or
```
select authmech(0)
```

**Example 3** Prints the authentication mechanism used for each login session:
```
select suid, authmech(spid)
from sysprocesses where suid!=0
```

**Usage**
- This function returns output of type varchar from one optional argument.
- If the value of the server process ID is 0, the function returns the authentication method used by the server process ID of the current client session.
- If no argument is specified, the output is the same as if the value of the server process ID is 0.
- Possible return values include ldap, ase, pam, and NULL.

**Permissions**
Any user can execute authmech to query a current personal session. You must have sso_role privileges to query the details of another user’s session.
biginttohex

Description

Returns the platform-independent 8 byte hexadecimal equivalent of the specified integer expression.

Syntax

biginttohex (integer_expression)

Parameters

integer_expression

is the integer value to be converted to a hexadecimal string.

Examples

Converts the big integer -922372036854775808 to a hexadecimal string:

1> select biginttohex(-9223372036854775808)
2> go

----------------
8000000000000000

Usage

• biginttohex, a datatype conversion function, returns the platform-independent hexadecimal equivalent of an integer, without a “0x” prefix.

• Use the biginttohex function for platform-independent conversions of integers to hexadecimal strings. biginttohex accepts any expression that evaluates to a bigint. It always returns the same hexadecimal equivalent for a given expression, regardless of the platform on which it is executed.

See also

Functions convert, hextobigint, hextoint, inttohex
**bintostr**

**Description**
Converts a sequence of hexadecimal digits to a string of its equivalent alphanumeric characters or varbinary data.

**Syntax**
```
select bintostr(sequence of hexadecimal digits)
```

**Parameters**
`sequence of hexadecimal digits` is the sequence of valid hexadecimal digits, consisting of \([0 – 9], [a – f] and [A – F]\), and which is prefixed with “0x”.

**Examples**
**Example 1** Converts the hexadecimal sequence of “0x723ad82fe” to an alphanumeric string of the same value:
```
1> select bintostr(0x723ad82fe)
2> go
-----------------------------------
0723ad82fe
```
In this example, the in-memory representation of the sequence of hexadecimal digits and its equivalent alphanumeric character string are:

<table>
<thead>
<tr>
<th>Hexadecimal digits (5 bytes)</th>
<th>Alphanumeric character string (9 bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 7 2 3 a d 8 2 f e</td>
<td>0 7 2 3 a d 8 2 f e</td>
</tr>
</tbody>
</table>

The function processes hexadecimal digits from right to left. In this example, the number of digits in the input is odd. For this reason, the alphanumeric character sequence has a prefix of “0” and is reflected in the output.

**Example 2** Converts the hexadecimal digits of a local variable called `@bin_data` to an alphanumeric string equivalent to the value of “723ad82fe”:
```
declare @bin_data varchar(30)
select @bin_data = 0x723ad82fe
select bintostr(@bin_data)
go
----------
0723ad82fe
```

**Usage**
- Any invalid characters in the input results in null as the output.
- The input must be valid varbinary data.
- A NULL input results in NULL output.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.
bintostr

Permissions
Any user can execute bintostr.

See also
Functions  strtobin
**cache_usage**

**Description**
Returns cache usage as a percentage of all objects in the cache to which the table belongs.

**Syntax**
cache_usage(table_name)

**Parameters**
- **table_name**
  is the name of a table. The name can be fully qualified (that is, it can include the database and owner name).

**Examples**

**Example 1** Returns percentage of the cache used by the titles tables:

```sql
select cache_usage("titles")
----------------------------
98.876953
```

**Example 2** Returns, from the master database, the percentage of the cache used by the authors tables:

```sql
select cache_usage ("pubs2..authors")
------------------------------------
98.876953
```

**Usage**
- `cache_usage` provides cache usage as percentage across all the pools of the cache.
- `cache_usage` does not provide any information on how much cache the current object is using, and does not provide information for cache usages of indexes if they are bound to different cache.
- In cluster environments – `cache_usage` provides cache usage of the cache the object is bound to in current node.

**Permissions**
Any user can execute `cache_usage`. 
CASE

Description
Supports conditional SQL expressions; can be used anywhere a value expression can be used.

Syntax
```
case
  when search_condition then expression
  [when search_condition then expression]...
  [else expression]
end
```

Parameters
```
case expression
  when expression then expression
  [when expression then expression]...
  [else expression]
end
```

Examples
**Example 1** Selects all the authors from the authors table and, for certain authors, specifies the city in which they live:
```
select au_lname, postalcode, 
case
  when postalcode = "94705"
  then "Berkeley Author"
  when postalcode = "94609"
  then "Oakland Author"
  when postalcode = "94612"
  then "Oakland Author"
```
when postalcode = "97330"
    then "Corvallis Author"
end
from authors

**Example 2**  Returns the first occurrence of a non-NULL value in either the `lowqty` or `highqty` column of the `discounts` table:

```sql
select stor_id, discount,
    coalesce (lowqty, highqty)
from discounts
```

You can also use the following format to produce the same result, since `coalesce` is an abbreviated form of a `case` expression:

```sql
select stor_id, discount,
    case
        when lowqty is not NULL then lowqty
        else highqty
    end
from discounts
```

**Example 3**  Selects the `titles` and `type` from the `titles` table. If the book type is `UNDECIDED`, `nullif` returns a NULL value:

```sql
select title,
    nullif(type, "UNDECIDED")
from titles
```

You can also use the following format to produce the same result, since `nullif` is an abbreviated form of a `case` expression:

```sql
select title,
    case
        when type = "UNDECIDED" then NULL
        else type
    end
from titles
```

**Example 4**  Produces an error message, because at least one expression must be something other than the null keyword:

```sql
select price, coalesce (NULL, NULL, NULL)
from titles
```

All result expressions in a `CASE` expression must not be NULL.

**Example 5**  Produces an error message, because at least two expressions must follow `coalesce`:

```sql
select stor_id, discount, coalesce (highqty) from discounts
```
A single coalesce element is illegal in a COALESCE expression.

Usage

- case expression simplifies standard SQL expressions by allowing you to express a search condition using a when...then construct instead of an if statement.

- case expressions can be used anywhere an expression can be used in SQL.

- If your query produces a variety of datatypes, the datatype of a case expression result is determined by datatype hierarchy, as described in “Datatypes of mixed-mode expressions” on page 7 in. If you specify two datatypes that Adaptive Server cannot implicitly convert (for example, char and int), the query fails.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
-case permission defaults to all users. No permission is required to use it.

See also
 Commands coalesce, nullif, if...else, select, where clause
 **cast**

**Description**

Returns the specified value, converted to another datatype. cast can change the nullability of the source expression, and uses the default format for date and time datatypes.

**Syntax**

```
cast (expression as datatype [(length | precision[, scale)])]
```

**Parameters**

- `expression` is the value to be converted from one datatype or date format to another. It includes columns, constants, functions, any combination of constants, and functions that are connected by arithmetic or bitwise operators or subqueries.

- When Java is enabled in the database, `expression` can be a value to be converted to a Java-SQL class.

- `unichar` is used as the destination datatype, the default length of 30 Unicode values is used if no length is specified.

- `length` is an optional parameter used with `char`, `nchar`, `unichar`, `univarchar`, `varchar`, `nvarchar`, `binary` and `varbinary` datatypes. If you do not supply a length, Adaptive Server truncates the data to 30 characters for character types and 30 bytes for binary types. The maximum allowable length for character and binary expression is 64K.

- `precision` is the number of significant digits in a numeric or decimal datatype. For float datatypes, precision is the number of significant binary digits in the mantissa. If you do not supply a precision, Adaptive Server uses the default precision of 18 for numeric and decimal datatypes.

- `scale` is the number of digits to the right of the decimal point in a numeric, or decimal datatype. If you do not supply a scale, Adaptive Server uses the default scale of 0.

**Examples**

**Example 1** Converts the date into a more readable `datetime` format:

```
select cast("01/03/63" as datetime)
go
```

```
-----------------------------------
Jan  3 1963 12:00AM
```

(1 row affected)

**Example 2** Converts the `total_sales` column in the `title` database to a 12-character column:
select title, cast(total_sales as char(12))

Usage

- For more information about datatype conversion, see “Datatype conversion functions” on page 63.
- `cast` generates a domain error when the argument falls outside the range over which the function is defined. This should happen rarely.
- Use `null` or `not null` to specify the nullability of a target column. You can use `null` or `not null` with `select into` to create a new table and change the datatype and nullability of existing columns in the source table.
- You can use `cast` to convert an image column to binary or varbinary. You are limited to the maximum length of the binary datatypes that is determined by the maximum column size for your server’s logical page size. If you do not specify the length, the converted value has a default length of 30 characters.
- You can use `unichar` expressions as a destination datatype, or they can be converted to another datatype. `unichar` expressions can be converted either explicitly between any other datatype supported by the server, or implicitly.
- If you do not specify length when `unichar` is used as a destination type, the default length of 30 Unicode values is used. If the length of the destination type is not large enough to accommodate the given expression, an error message appears.

Implicit conversion

Implicit conversion between types when the primary fields do not match may cause data truncation, the insertion of a default value, or an error message to be raised. For example, when a `datetime` value is converted to a `date` value, the time portion is truncated, leaving only the date portion. If a time value is converted to a `datetime` value, a default date portion of Jan 1, 1900 is added to the new `datetime` value. If a `date` value is converted to a `datetime` value, a default time portion of 00:00:00:000 is added to the `datetime` value.

- `DATE` -> `VARCHAR`, `CHAR`, `BINARY`, `VARBINARY`, `DATETIME`, `SMALLDATETIME`
- `TIME` -> `VARCHAR`, `CHAR`, `BINARY`, `VARBINARY`, `DATETIME`, `SMALLDATETIME`
- `VARCHAR`, `CHAR`, `BINARY`, `VARBINARY`, `DATETIME`, `SMALLDATETIME` -> `DATE`
- `VARCHAR`, `CHAR`, `BINARY`, `VARBINARY`, `DATETIME`, `SMALLDATETIME` -> `TIME`

Explicit conversion

If you attempt to explicitly convert a `date` to a `datetime`, and the value is outside the `datetime` range such as “Jan 1, 1000” the conversion is not allowed and an informative error message is raised.

- `DATE` -> `UNICHAR`, `UNIVARCHAR`
TIME -> UNICHAR, UNIVARCHAR
UNICHAR, UNIVARCHAR -> DATE
UNICHAR, UNIVARCHAR -> TIME

Conversions involving Java classes

- When Java is enabled in the database, you can use `cast` to change datatypes in these ways:
  - Convert Java object types to SQL datatypes.
  - Convert SQL datatypes to Java types.
  - Convert any Java-SQL class installed in Adaptive Server to any other Java-SQL class installed in Adaptive Server if the compile-time datatype of the expression (the source class) is a subclass or superclass of the target class.

The result of the conversion is associated with the current database.

Standards  ANSI SQL – Compliance level: ANSI compliant.
Permissions Any user can execute `cast`.
ceiling

Description
Returns the smallest integer greater than or equal to the specified value.

Syntax
ceiling(value)

Parameters
value
is a column, variable, or expression with a datatype is exact numeric,
approximate numeric, money, or any type that can be implicitly converted
to one of these types.

Examples

Example 1 Returns a value of 124:
select ceiling(123.45)
124

Example 2 Returns a value of -123:
select ceiling(-123.45)
-123

Example 3 Returns a value of 24.000000:
select ceiling(1.2345E2)
24.000000

Example 4 Returns a value of -123.000000:
select ceiling(-1.2345E2)
-123.000000

Example 5 Returns a value of 124.00
select ceiling($123.45)
124.00

Example 6 Returns values of “discount” from the salesdetail table where
title_id is the value “PS3333”:
select discount, ceiling(discount) from salesdetail
where title_id = "PS3333"

+---------------------+---------------------+
| discount            | ceiling(discount)   |
+---------------------+---------------------+
| 45.000000           | 45.000000           |
| 46.700000           | 47.000000           |
| 46.700000           | 47.000000           |
| 50.000000           | 50.000000           |
Usage
• ceiling, a mathematical function, returns the smallest integer that is greater than or equal to the specified value. The return value has the same datatype as the value supplied.

For numeric and decimal values, results have the same precision as the value supplied and a scale of zero.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute ceiling.

See also
For general information about mathematical functions, see “Mathematical functions” on page 74.

Command  set
Functions  abs, floor, round, sign
char

Description
Returns the character equivalent of an integer.

Syntax
char(integer_expr)

Parameters
integer_expr
is any integer (tinyint, smallint, or int) column name, variable, or constant expression between 0 and 255.

Examples

Example 1

select char(42)
- *

Example 2

select xxx = char(65)
xxx
---
A

Usage

- char, a string function, converts a single-byte integer value to a character value (char is usually used as the inverse of ascii).

- char returns a char datatype. If the resulting value is the first byte of a multibyte character, the character may be undefined.

- If char_expr is NULL, returns NULL.

Reformatting output with char

- You can use concatenation and char values to add tabs or carriage returns to reformat output. char(10) converts to a return; char(9) converts to a tab. For example:

/* just a space */
select title_id + " " + title from titles where title_id = "T67061"

/* a return */
select title_id + char(10) + title from titles where title_id = "T67061"

/* a tab */
select title_id + char(9) + title from titles where title_id = "T67061"

-----------------------------------------------------------------------
T67061 Programming with Curses
-----------------------------------------------------------------------
T67061

Programming with Curses
Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute char.

See also

For general information about string functions, see “String functions” on page 76.

Functions

ascii, str
**char_length**

Description

Returns the number of characters in an expression.

Syntax

```sql
char_length(char_expr | uchar_expr)
```

Parameters

- **char_expr**
  - is a character-type column name, variable, or constant expression of `char`, `varchar`, `nchar`, or `nvarchar` type.

- **uchar_expr**
  - is a character-type column name, variable, or constant expression of `unichar` or `univarchar` type.

Examples

**Example 1**

```sql
select char_length(notes) from titles
where title_id = "PC9999"
```

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>39</td>
</tr>
</tbody>
</table>

**Example 2**

```sql
declare @var1 varchar(20), @var2 varchar(20), @char char(20)
select @var1 = "abcd", @var2 = "abcd  ", @char = "abcd"
select char_length(@var1), char_length(@var2), char_length(@char)
```

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>8</td>
<td>20</td>
</tr>
</tbody>
</table>

Usage

- `char_length`, a string function, returns an integer representing the number of characters in a character expression or text value.

- For variable-length columns and variables, `char_length` returns the number of characters (not the defined length of the column or variable). If explicit trailing blanks are included in variable-length variables, they are not stripped. For literals and fixed-length character columns and variables, `char_length` does not strip the expression of trailing blanks (see Example 2).

- For `unitext`, `unichar`, and `univarchar` columns, `char_length` returns the number of Unicode values (16-bit), with one surrogate pair counted as two Unicode values. For example, this is what is returned if a `unitext` column `ut` contains row value `U+0041U+0042U+d800dc00`:

```sql
select char_length(ut) from unitable
```

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
</tr>
</tbody>
</table>
For multibyte character sets, the number of characters in the expression is usually fewer than the number of bytes; use `datalength` to determine the number of bytes.

For Unicode expressions, returns the number of Unicode values (not bytes) in an expression. Surrogate pairs count as two Unicode values.

If `char_expr` or `uchar_expr` is NULL, `char_length` returns NULL.

For general information about string functions, see “String functions” on page 76.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute `char_length`.

**See also**

**Function** `datalength`
charindex

Description

Returns an integer representing the starting position of an expression.

Syntax

charindex(expression1, expression2)

Parameters

expression

is a binary or character column name, variable, or constant expression. Can be char, varchar, nchar, nvarchar, unichar or univarchar, binary, or varbinary.

Examples

Returns the position at which the character expression “wonderful” begins in the notes column of the titles table:

```sql
select charindex("wonderful", notes)
from titles
where title_id = "TC3218"
 -----------
    46
```

Usage

- charindex, a string function, searches expression2 for the first occurrence of expression1 and returns an integer representing its starting position. If expression1 is not found, charindex returns 0.
- If expression1 contains wildcard characters, charindex treats them as literals.
- If expression2 is NULL, returns 0.
- If a varchar expression is given as one parameter and a unichar expression as the other, the varchar expression is implicitly converted to unichar (with possible truncation).

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute charindex.

See also

For general information about string functions, see “String functions” on page 76.

Function

patindex
coalesce

Description
Supports conditional SQL expressions; can be used anywhere a value expression can be used; alternative for a case expression.

Syntax
\texttt{coalesce(expression, expression [\ texttt{, expression}]...)}

Parameters
- \texttt{coalesce} evaluates the listed expressions and returns the first non-null value. If all expressions are null, \texttt{coalesce} returns NULL.
- \texttt{expression} is a column name, a constant, a function, a subquery, or any combination of column names, constants, and functions connected by arithmetic or bitwise operators. For more information about expressions, see “Expressions” on page 359.

Examples
- **Example 1** Returns the first occurrence of a non-null value in either the \texttt{lowqty} or \texttt{highqty} column of the \texttt{discounts} table:
  \begin{verbatim}
  select stor_id, discount, coalesce (lowqty, highqty)
  from discounts
  \end{verbatim}

- **Example 2** An alternative way of writing the previous example:
  \begin{verbatim}
  select stor_id, discount, 
  case
    when lowqty is not NULL then lowqty
    else highqty
  end
  from discounts
  \end{verbatim}

Usage
- \texttt{coalesce} expression simplifies standard SQL expressions by allowing you to express a search condition as a simple comparison instead of using a when...then construct.
- You can use \texttt{coalesce} expressions anywhere an expression in SQL.
- At least one result of the \texttt{coalesce} expression must return a non-null value. This example produces the following error message:
  \begin{verbatim}
  select price, coalesce (NULL, NULL, NULL)
  from titles
  \end{verbatim}

All result expressions in a CASE expression must not be NULL.
If your query produces a variety of datatypes, the datatype of a case expression result is determined by datatype hierarchy, as described in “Datatypes of mixed-mode expressions” on page 7. If you specify two datatypes that Adaptive Server cannot implicitly convert (for example, char and int), the query fails.

**coalesce** is an abbreviated form of a case expression. Example 2 describes an alternative way of writing the coalesce statement.

**coalesce** must be followed by at least two expressions. This example produces the following error message:

```sql
select stor_id, discount, coalesce (highqty)
from discounts
```

A single coalesce element is illegal in a COALESCE expression.

**Standards**  
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**  
Any user can execute coalesce.

**See also**  
Commands: case, nullif, select, if...else, where clause
### col_length

**Description**
Returns the defined length of a column.

**Syntax**
```
col_length(object_name, column_name)
```

**Parameters**
- `object_name` is name of a database object, such as a table, view, procedure, trigger, default, or rule. The name can be fully qualified (that is, it can include the database and owner name). It must be enclosed in quotes.
- `column_name` is the name of the column.

**Examples**
Finds the length of the `title` column in the `titles` table. The “x” gives a column heading to the result:
```
select x = col_length("titles", "title")
x
---
80
```

**Usage**
- `col_length`, a system function, returns the defined length of column.
- For general information about system functions, see “System functions” on page 77.
- To find the actual length of the data stored in each row, use `datalength`.
- For text, `unitext`, and `image` columns, `col_length` returns 16, the length of the `binary(16)` pointer to the actual text page.
- For `unichar` columns, the defined length is the number of Unicode values declared when the column was defined (not the number of bytes represented).

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute `col_length`.

**See also**
- **Function** `datalength`
col_name

Description

Returns the name of the column where the table and column IDs are specified, and can be up to 255 bytes in length.

Syntax

```
col_name(object_id, column_id [, database_id])
```

Parameters

- **object_id**
  - is a numeric expression that is an object ID for a table, view, or other database object. These are stored in the id column of `sysobjects`.

- **column_id**
  - is a numeric expression that is a column ID of a column. These are stored in the colid column of `syscolumns`.

- **database_id**
  - is a numeric expression that is the ID for a database. These are stored in the db_id column of `sysdatabases`.

Examples

```
select col_name(208003772, 2)
```

Usage

- `col_name`, a system function, returns the column’s name.
- For general information about system functions, see “System functions” on page 77.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute `col_name`.

See also

`Functions`  `db_id, object_id`
**compare**

**Description**
Allows you to directly compare two character strings based on alternate collation rules.

**Syntax**
```
compare (%char_expression1|uchar_expression1),
  (%char_expression2|uchar_expression2),
  [%collation_name | collation_ID]
```

**Parameters**
- `char_expression1` or `uchar_expression1` are the character expressions to compare to `char_expression2` or `uchar_expression2`.
- `char_expression2` or `uchar_expression2` are the character expressions against which to compare `char_expression1` or `uchar_expression1`.
- `char_expression1` and `char_expression2` can be:
  - Character type (char, varchar, nchar, or nvarchar)
  - Character variable, or
  - Constant character expression, enclosed in single or double quotation marks
- `uchar_expression1` and `uchar_expression2` can be:
  - Character type (unichar or univarchar)
  - Character variable, or
  - Constant character expression, enclosed in single or double quotation marks
- `collation_name` can be a quoted string or a character variable that specifies the collation to use. Table 2-6 on page 116 shows the valid values.
- `collation_ID` is an integer constant or a variable that specifies the collation to use. Table 2-6 on page 116 shows the valid values.

**Examples**
**Example 1** Compares `aaa` and `bbb`:
```
1> select compare ("aaa","bbb")
2> go

-----------
-1
(1 row affected)
```
Alternatively, you can also compare `aaa` and `bbb` using this format:

```sql
1> select compare (('aaa'),('bbb'))
2> go

-----------
-1
(1 row affected)
```

**Example 2** Compares `aaa` and `bbb` and specifies binary sort order:

```sql
1> select compare ('aaa','bbb','binary')
2> go

-----------
-1
(1 row affected)
```

Alternatively, you can compare `aaa` and `bbb` using this format, and the collation ID instead of the collation name:

```sql
1> select compare (('aaa'),('bbb'),(50))
2> go

-----------
-1
(1 row affected)
```

**Usage**

- The `compare` function returns the following values, based on the collation rules that you chose:
  - 1 – indicates that `char_expression1` or `uchar_expression1` is greater than `char_expression2` or `uchar_expression2`.
  - 0 – indicates that `char_expression1` or `uchar_expression1` is equal to `char_expression2` or `uchar_expression2`.
  - -1 – indicates that `char_expression1` or `uchar_expression1` is less than `char_expression2` or `uchar_expression2`.

- `compare` can generate up to six bytes of collation information for each input character. Therefore, the result from using `compare` may exceed the length limit of the `varbinary` datatype. If this happens, the result is truncated to fit. Adaptive Server issues a warning message, but the query or transaction that contained the `compare` function continues to run. Since this limit is dependent on the logical page size of your server, truncation removes result bytes for each input character until the result string is less than the following for DOL and APL tables:
Table 2-5: Maximum row and column length—APL and DOL

<table>
<thead>
<tr>
<th>Locking scheme</th>
<th>Page size</th>
<th>Maximum row length</th>
<th>Maximum column length</th>
</tr>
</thead>
<tbody>
<tr>
<td>APL tables</td>
<td>2K (2048 bytes)</td>
<td>1962</td>
<td>1960 bytes</td>
</tr>
<tr>
<td></td>
<td>4K (4096 bytes)</td>
<td>4010</td>
<td>4008 bytes</td>
</tr>
<tr>
<td></td>
<td>8K (8192 bytes)</td>
<td>8106</td>
<td>8104 bytes</td>
</tr>
<tr>
<td></td>
<td>16K (16384 bytes)</td>
<td>16298</td>
<td>16296 bytes</td>
</tr>
<tr>
<td>DOL tables</td>
<td>2K (2048 bytes)</td>
<td>1964</td>
<td>1958 bytes</td>
</tr>
<tr>
<td></td>
<td>4K (4096 bytes)</td>
<td>4012</td>
<td>4006 bytes</td>
</tr>
<tr>
<td></td>
<td>8K (8192 bytes)</td>
<td>8108</td>
<td>8102 bytes</td>
</tr>
<tr>
<td></td>
<td>16K (16384 bytes)</td>
<td>16300</td>
<td>16294 bytes if table does not include any variable length columns</td>
</tr>
<tr>
<td></td>
<td>16K (16384 bytes)</td>
<td>16300 (subject to a max start offset of varlen = 8191)</td>
<td>8191-6-2 = 8183 bytes if table includes at least on variable length column.*</td>
</tr>
</tbody>
</table>

* This size includes six bytes for the row overhead and two bytes for the row length field

- Both `char_expression1`, `uchar_expression1`, and `char_expression2`, `uchar_expression2` must be characters that are encoded in the server’s default character set.
- `char_expression1`, `uchar_expression 1`, or `char_expression2`, `uchar_expression2`, or both, can be empty strings:
  - If `char_expression2` or `uchar_expression2` is empty, the function returns 1.
  - If both strings are empty, then they are equal, and the function returns 0.
  - If `char_expression1` or `uchar_expression 1` is empty, the function returns -1.

The `compare` function does not equate empty strings and strings containing only spaces. `compare` uses the `sortkey` function to generate collation keys for comparison. Therefore, a truly empty string, a string with one space, or a string with two spaces do not compare equally.

- If either `char_expression1`, `uchar_expression1`; or `char_expression2`, `uchar_expression2` is NULL, then the result is NULL.
- If a `varchar` expression is given as one parameter and a `unichar` expression is given as the other, the `varchar` expression is implicitly converted to `unichar` (with possible truncation).
- If you do not specify a value for `collation_name` or `collation_ID`, `compare` assumes binary collation.
Table 2-6 lists the valid values for `collation_name` and `collation_ID`.

<table>
<thead>
<tr>
<th>Description</th>
<th>Collation name</th>
<th>Collation ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deafult Unicode multilingual</td>
<td>default</td>
<td>20</td>
</tr>
<tr>
<td>Thai dictionary order</td>
<td>thaidict</td>
<td>21</td>
</tr>
<tr>
<td>ISO14651 standard</td>
<td>iso14651</td>
<td>22</td>
</tr>
<tr>
<td>UTF-16 ordering – matches UTF-8 binary ordering</td>
<td>utf8bin</td>
<td>24</td>
</tr>
<tr>
<td>CP 850 Alternative – no accent</td>
<td>altnoacc</td>
<td>39</td>
</tr>
<tr>
<td>CP 850 Alternative – lowercase first</td>
<td>altdict</td>
<td>45</td>
</tr>
<tr>
<td>CP 850 Western European – no case preference</td>
<td>altnocsp</td>
<td>46</td>
</tr>
<tr>
<td>CP 850 Scandinavian – dictionary ordering</td>
<td>scandict</td>
<td>47</td>
</tr>
<tr>
<td>CP 850 Scandinavian – case-insensitive with preference</td>
<td>scannocp</td>
<td>48</td>
</tr>
<tr>
<td>GB Pinyin</td>
<td>gbpinyin</td>
<td>n/a</td>
</tr>
<tr>
<td>Binary sort</td>
<td>binary</td>
<td>50</td>
</tr>
<tr>
<td>Latin-1 English, French, German dictionary</td>
<td>dict</td>
<td>51</td>
</tr>
<tr>
<td>Latin-1 English, French, German no case</td>
<td>nocase</td>
<td>52</td>
</tr>
<tr>
<td>Latin-1 English, French, German no case, preference</td>
<td>nocasep</td>
<td>53</td>
</tr>
<tr>
<td>Latin-1 English, French, German no accent</td>
<td>noaccent</td>
<td>54</td>
</tr>
<tr>
<td>Latin-1 Spanish dictionary</td>
<td>espdict</td>
<td>55</td>
</tr>
<tr>
<td>Latin-1 Spanish no case</td>
<td>espnocs</td>
<td>56</td>
</tr>
<tr>
<td>Latin-1 Spanish no accent</td>
<td>espnoac</td>
<td>57</td>
</tr>
<tr>
<td>ISO 8859-5 Russian dictionary</td>
<td>rusdict</td>
<td>58</td>
</tr>
<tr>
<td>ISO 8859-5 Russian no case</td>
<td>rusnocs</td>
<td>59</td>
</tr>
<tr>
<td>ISO 8859-5 Cyrillic dictionary</td>
<td>cyrdict</td>
<td>63</td>
</tr>
<tr>
<td>ISO 8859-5 Cyrillic no case</td>
<td>cyrnocs</td>
<td>64</td>
</tr>
<tr>
<td>ISO 8859-7 Greek dictionary</td>
<td>elldict</td>
<td>65</td>
</tr>
<tr>
<td>ISO 8859-2 Hungarian dictionary</td>
<td>hundict</td>
<td>69</td>
</tr>
<tr>
<td>ISO 8859-2 Hungarian no accents</td>
<td>hunnoac</td>
<td>70</td>
</tr>
<tr>
<td>ISO 8859-2 Hungarian no case</td>
<td>hunnocs</td>
<td>71</td>
</tr>
<tr>
<td>ISO 8859-9 Turkish dictionary</td>
<td>turdict</td>
<td>72</td>
</tr>
<tr>
<td>ISO 8859-9 Turkish no accents</td>
<td>turknoac</td>
<td>73</td>
</tr>
<tr>
<td>ISO 8859-9 Turkish no case</td>
<td>turknocs</td>
<td>74</td>
</tr>
<tr>
<td>CP932 binary ordering</td>
<td>cp932bin</td>
<td>129</td>
</tr>
<tr>
<td>Chinese phonetic ordering</td>
<td>dynix</td>
<td>130</td>
</tr>
<tr>
<td>GB2312 binary ordering</td>
<td>gb2312bn</td>
<td>137</td>
</tr>
<tr>
<td>Common Cyrillic dictionary</td>
<td>cyrdict</td>
<td>140</td>
</tr>
<tr>
<td>Turkish dictionary</td>
<td>turdict</td>
<td>155</td>
</tr>
<tr>
<td>Description</td>
<td>Collation name</td>
<td>Collation ID</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>----------------</td>
<td>--------------</td>
</tr>
<tr>
<td>EUCKSC binary ordering</td>
<td>euckscbn</td>
<td>161</td>
</tr>
<tr>
<td>Chinese phonetic ordering</td>
<td>gbpinyn</td>
<td>163</td>
</tr>
<tr>
<td>Russian dictionary ordering</td>
<td>rusdict</td>
<td>165</td>
</tr>
<tr>
<td>SJIS binary ordering</td>
<td>sjisbin</td>
<td>179</td>
</tr>
<tr>
<td>EUCJIS binary ordering</td>
<td>eucjisbn</td>
<td>192</td>
</tr>
<tr>
<td>BIG5 binary ordering</td>
<td>big5bin</td>
<td>194</td>
</tr>
<tr>
<td>Shift-JIS binary order</td>
<td>sjisbin</td>
<td>259</td>
</tr>
</tbody>
</table>

Standards: ANSI SQL – Compliance level: Transact-SQL extension.

Permissions: Any user can execute compare.

See also: Function sortkey
**convert**

Description

Returns the specified value, converted to another datatype or a different datetime display format.

Syntax

```plaintext
convert (datatype [(length) | (precision, scale)])
    [null | not null, expression [, style]]
```

Parameters

- **datatype**
  - is the system-supplied datatype (for example, `char(10)`, `unichar (10)`, `varbinary (50)`, or `int`) into which to convert the expression. You cannot use user-defined datatypes.
  - When Java is enabled in the database, *datatype* can also be a Java-SQL class in the current database.

- **length**
  - is an optional parameter used with `char`, `nchar`, `unichar`, `univarchar`, `varchar`, `nvarchar`, `binary`, and `varbinary` datatypes. If you do not supply a length, Adaptive Server truncates the data to 30 characters for the character types and 30 bytes for the binary types. The maximum allowable length for character and binary expression is 64K.

- **precision**
  - is the number of significant digits in a numeric or decimal datatype. For float datatypes, precision is the number of significant binary digits in the mantissa. If you do not supply a precision, Adaptive Server uses the default precision of 18 for numeric and decimal datatypes.

- **scale**
  - is the number of digits to the right of the decimal point in a numeric, or decimal datatype. If you do not supply a scale, Adaptive Server uses the default scale of 0.

- **null | not null**
  - specifies the nullability of the result expression. If you do not supply either `null` or `not null`, the converted result has the same nullability as the expression.

- **expression**
  - is the value to be converted from one datatype or date format to another.
  - When Java is enabled in the database, *expression* can be a value to be converted to a Java-SQL class.
  - When `unichar` is used as the destination datatype, the default length of 30 Unicode values is used if no length is specified.
**style**

is the display format to use for the converted data. When converting `money` or `smallmoney` data to a character type, use a `style` of 1 to display a comma after every 3 digits.

When converting `datetime` or `smalldatetime` data to a character type, use the style numbers in Table 2-7 to specify the display format. Values in the left-most column display 2-digit years (yy). For 4-digit years (yyyy), add 100, or use the value in the middle column.

When converting `date` data to a character type, use style numbers 1 through 7 (101 through 107) or 10 through 12 (110 through 112) in Table 2-7 to specify the display format. The default value is 100 (`mon dd yyyy hh:mmAM` (or PM)). If `date` data is converted to a style that contains a time portion, that time portion reflects the default value of zero.

When converting `time` data to a character type, use style number 8 or 9 (108 or 109) to specify the display format. The default is 100 (`mon dd yyyy hh:mmAM` (or PM)). If `time` data is converted to a style that contains a date portion, the default date of Jan 1, 1900 is displayed.

**Table 2-7: Date format conversions using the style parameter**

<table>
<thead>
<tr>
<th>Without century (yy)</th>
<th>With century (yyyy)</th>
<th>Standard</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0 or 100</td>
<td>Default</td>
<td><code>mon dd yyyy hh:mm AM</code> (or PM)</td>
</tr>
<tr>
<td>1</td>
<td>101</td>
<td>USA</td>
<td><code>mm/dd/yy</code></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>SQL standard</td>
<td><code>yy/mm.dd</code></td>
</tr>
<tr>
<td>3</td>
<td>103</td>
<td>English/French</td>
<td><code>dd/mm/yy</code></td>
</tr>
<tr>
<td>4</td>
<td>104</td>
<td>German</td>
<td><code>dd.mm.yy</code></td>
</tr>
<tr>
<td>5</td>
<td>105</td>
<td></td>
<td><code>dd-mm-yy</code></td>
</tr>
<tr>
<td>6</td>
<td>106</td>
<td></td>
<td><code>dd mon yy</code></td>
</tr>
<tr>
<td>7</td>
<td>107</td>
<td></td>
<td><code>mon dd, yy</code></td>
</tr>
<tr>
<td>8</td>
<td>108</td>
<td></td>
<td><code>HH:mm:ss</code></td>
</tr>
<tr>
<td>-</td>
<td>9 or 109</td>
<td>Default + milliseconds</td>
<td><code>mon dd yyyy hh:mm:ss AM</code> (or PM)</td>
</tr>
<tr>
<td>10</td>
<td>110</td>
<td>USA</td>
<td><code>mm-dd-yy</code></td>
</tr>
<tr>
<td>11</td>
<td>111</td>
<td>Japan</td>
<td><code>yy/mm/dd</code></td>
</tr>
<tr>
<td>12</td>
<td>112</td>
<td>ISO</td>
<td><code>ymmdd</code></td>
</tr>
<tr>
<td>13</td>
<td>113</td>
<td></td>
<td><code>yy/dd/mm</code></td>
</tr>
<tr>
<td>14</td>
<td>114</td>
<td></td>
<td><code>mm/yy/dd</code></td>
</tr>
</tbody>
</table>

**Key**  
“mon” indicates a month spelled out, “mm” the month number or minutes. “HH” indicates a 24-hour clock value, “hh” a 12-hour clock value. The last row, 23, includes a literal “T” to separate the date and time portions of the format.
The default values (style 0 or 100), and style 9 or 109 return the century (yyyy). When converting to char or varchar from smalldatetime, styles that include seconds or milliseconds show zeros in those positions.

<table>
<thead>
<tr>
<th>Without century (yy)</th>
<th>With century (yyyy)</th>
<th>Standard</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>14 114</td>
<td></td>
<td>hh:mm:ss:mmmAM (or PM)</td>
<td></td>
</tr>
<tr>
<td>15 115</td>
<td></td>
<td>dd/yy/mm</td>
<td></td>
</tr>
<tr>
<td>- 16 or 116</td>
<td></td>
<td>mon dd yyyy HH:mm:ss</td>
<td></td>
</tr>
<tr>
<td>17 117</td>
<td></td>
<td>hh:mmAM</td>
<td></td>
</tr>
<tr>
<td>18 118</td>
<td></td>
<td>HH:mm</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td>hh:mm:ss:zzzAM</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td>hh:mm:ss:zzz</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td>yy/mm/dd HH:mm:ss</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td>yy/mm/dd HH:mm AM (or PM)</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td>yyyy-mm-ddTHH:mm:ss</td>
<td></td>
</tr>
</tbody>
</table>

**Key**  “mon” indicates a month spelled out, “mm” the month number or minutes. “HH” indicates a 24-hour clock value, “hh” a 12-hour clock value. The last row, 23, includes a literal “T” to separate the date and time portions of the format.

The default values (style 0 or 100), and style 9 or 109 return the century (yyyy). When converting to char or varchar from smalldatetime, styles that include seconds or milliseconds show zeros in those positions.

**Examples**

**Example 1**

```sql
select title, convert(char(12), total_sales)
from titles
```

**Example 2**

```sql
select title, total_sales
from titles
where convert(char(20), total_sales) like "1%"
```

**Example 3** Converts the current date to style 3, dd/mm/yy:

```sql
select convert(char(12), getdate(), 3)
```

**Example 4** If the value pubdate can be null, you must use varchar rather than char, or errors may result:

```sql
select convert(varchar(12), pubdate, 3) from titles
```

**Example 5** Returns the integer equivalent of the string “0x00000100”. Results can vary from one platform to another:

```sql
select convert(integer, 0x00000100)
```

**Example 6** Returns the platform-specific bit pattern as a Sybase binary type:

```sql
select convert (binary, 10)
```
Example 7  Returns 1, the bit string equivalent of $.11:

```sql
select convert(bit, $.11)
```

Example 8  Creates #tempsales with total_sales of datatype char(100), and does not allow null values. Even if titles.total_sales was defined as allowing nulls, #tempsales is created with #tempsales.total_sales not allowing null values:

```sql
select title, convert (char(100) not null, total_sales) into #tempsales
from titles
```

Usage

- `convert`, a datatype conversion function, converts between a wide variety of datatypes and reformats date/time and money data for display purposes.
- For more information about datatype conversion, see “Datatype conversion functions” on page 63.
- `convert` – returns the specified value, converted to another datatype or a different datetime display format. When converting from untext to other character and binary datatypes, the result is limited to the maximum length of the destination datatype. If the length is not specified, the converted value has a default size of 30 bytes. If you are using `enable surrogate processing`, a surrogate pair is returned as a whole. For example, this is what is returned if you convert a untext column that contains data U+0041U+0042U+20acU+0043 (stands for “AB €”) to a UTF-8 varchar(3) column:

```sql
select convert(varchar(3), ut) from untable
```

```sql
---
AB
```

- `convert` generates a domain error when the argument falls outside the range over which the function is defined. This should happen rarely.
- Use `null` or `not null` to specify the nullability of a target column. Specifically, this can be used with `select into` to create a new table and change the datatype and nullability of existing columns in the source table (See Example 8, above).

The result is an undefined value if:

- The expression being converted is to a `not null` result.
- The expression’s value is null.

Use the following `select` statement to generate a known non-NULL value for predictable results:

```sql
select convert(int not null isnull(col2, 5)) from table1
```
You can use `convert` to convert an image column to binary or varbinary. You are limited to the maximum length of the binary datatypes, which is determined by the maximum column size for your server’s logical page size. If you do not specify the length, the converted value has a default length of 30 characters.

You can use `unichar` expressions as a destination datatype or you can convert them to another datatype. `unichar` expressions can be converted either explicitly between any other datatype supported by the server, or implicitly.

If you do not specify the length when `unichar` is used as a destination type, the default length of 30 Unicode values is used. If the length of the destination type is not large enough to accommodate the given expression, an error message appears.

**Implicit conversion**

Implicit conversion between types when the primary fields do not match may cause data truncation, the insertion of a default value, or an error message to be raised. For example, when a `datetime` value is converted to a `date` value, the time portion is truncated, leaving only the date portion. If a `time` value is converted to a `datetime` value, a default date portion of Jan 1, 1900 is added to the new `datetime` value. If a `date` value is converted to a `datetime` value, a default time portion of 00:00:00:000 is added to the `datetime` value.

```
DATE -> VARCHAR, CHAR, BINARY, VARBINARY, DATETIME, SMALLDATETIME
TIME -> VARCHAR, CHAR, BINARY, VARBINARY, DATETIME, SMALLDATETIME
VARCHAR, CHAR, BINARY, VARBINARY, DATETIME, SMALLDATETIME -> DATE
VARCHAR, CHAR, BINARY, VARBINARY, DATETIME, SMALLDATETIME -> TIME
```

**Explicit conversion**

If you attempt to explicitly convert a `date` to a `datetime` and the value is outside the `datetime` range, such as “Jan 1, 1000” the conversion is not allowed and an informative error message is raised.

```
DATE -> UNICHAR, UNIVARCHAR
TIME -> UNICHAR, UNIVARCHAR
UNICHAR, UNIVARCHAR -> DATE
UNICHAR, UNIVARCHAR -> TIME
```

**Conversions involving Java classes**

- When Java is enabled in the database, you can use `convert` to change datatypes in these ways:
  - Convert Java object types to SQL datatypes.
  - Convert SQL datatypes to Java types.
• Convert any Java-SQL class installed in Adaptive Server to any other Java-SQL class installed in Adaptive Server if the compile-time datatype of the expression (the source class) is a subclass or superclass of the target class.

The result of the conversion is associated with the current database.

**Standards**

ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**

Any user can execute `convert`.

**See also**

Documents *Java in Adaptive Server Enterprise* for a list of allowed datatype mappings and more information about datatype conversions involving Java classes.

**Datatypes**

User-defined datatypes

**Functions**

`hextoint`, `inttohex`
COS

Description  Returns the cosine of the specified angle.

Syntax  cos(angle)

Parameters  angle

is any approximate numeric (float, real, or double precision) column name,
variable, or constant expression.

Examples  select cos(44)

0.999843

Usage  • cos, a mathematical function, returns the cosine of the specified angle, in
radians.

• For general information about mathematical functions, see “Mathematical
functions” on page 74.

Standards  ANSI SQL – Compliance level: Transact-SQL extension.

Permissions  Any user can execute cos.

See also  Functions  acos, degrees, radians, sin
cot

Description
Returns the cotangent of the specified angle.

Syntax
cot(angle)

Parameters
angle
is any approximate numeric (float, real, or double precision) column name,
variable, or constant expression.

Examples
select cot(90)
--------------------
-0.501203

Usage
• cot, a mathematical function, returns the cotangent of the specified angle,
in radians.

• For general information about mathematical functions, see “Mathematical
functions” on page 74.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute cot.

See also
Functions degrees, radians, sin
**count**

**Description**
Returns the number of (distinct) non-null values, or the number of selected rows as an integer.

**Syntax**
```
count([all | distinct] expression)
```

**Parameters**
- `all` applies `count` to all values. `all` is the default.
- `distinct` eliminates duplicate values before `count` is applied. `distinct` is optional.
- `expression` is a column name, constant, function, any combination of column names, constants, and functions connected by arithmetic or bitwise operators, or a subquery. With aggregates, an expression is usually a column name. For more information, see “Expressions” on page 359.

**Examples**

**Example 1** Finds the number of different cities in which authors live:
```
select count(distinct city)
from authors
```

**Example 2** Lists the types in the `titles` table, but eliminates the types that include only one book or none:
```
select type
from titles
group by type
having count(*) > 1
```

**Usage**
- `count`, an aggregate function, finds the number of non-null values in a column. For general information about aggregate functions, see “Aggregate functions” on page 54.
- When `distinct` is specified, `count` finds the number of unique non-null values. `count` can be used with all datatypes, including `unichar`, but cannot be used with `text` and `image`. Null values are ignored when counting.
- `count(column_name)` returns a value of 0 on empty tables, on columns that contain only null values, and on groups that contain only null values.
- `count(*)` finds the number of rows. `count(*)` does not take any arguments, and cannot be used with `distinct`. All rows are counted, regardless of the presence of null values.
• When tables are being joined, include `count(*)` in the **select list** to produce the count of the number of rows in the joined results. If the objective is to count the number of rows from one table that match criteria, use `count(column_name)`.

• You can use `count` as an existence check in a subquery. For example:

```sql
select * from tab where 0 <
  (select count(*) from tab2 where ...)
```

However, because `count` counts all matching values, `exists` or `in` may return results faster. For example:

```sql
select * from tab where exists
  (select * from tab2 where ...)
```

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute `count`.

**See also**
- **Commands**
  - compute clause, group by and having clauses, select, where clause
count_big

Description
Returns the number of (distinct) non-null values or the number of selected rows as a bigint.

Syntax
count_big([all | distinct] expression)

Parameters
all
   applies count_big to all values. all is the default.

distinct
   eliminates duplicate values before count_big is applied. distinct is optional.

expression
   is a column name, constant, function, any combination of column names, constants, and functions connected by arithmetic or bitwise operators, or a subquery. With aggregates, an expression is usually a column name.

Examples
Finds the number of occurrences of name in systypes:

1> select count_big(name) from systypes  
2> go
---------------------
        42

Usage
• count_big, an aggregate function, finds the number of non-null values in a column.

• When distinct is specified, count_big finds the number of unique non-null values. Null values are ignored when counting.

• count_big(column_name) returns a value of 0 on empty tables, on columns that contain only null values, and on groups that contain only null values.

• count_big(*) finds the number of rows. count_big(*) does not take any arguments, and cannot be used with distinct. All rows are counted, regardless of the presence of null values.

• When tables are being joined, include count_big(*) in the select list to produce the count of the number of rows in the joined results. If the objective is to count the number of rows from one table that match criteria, use count_big(column_name).

• You can use count_big as an existence check in a subquery. For example:

   select * from tab where 0 <
      (select count_big(*) from tab2 where ...)

   However, because count_big counts all matching values, exists or in may return results faster. For example:
select * from tab where exists
  (select * from tab2 where ...)

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute count_big.

See also
Commands compute clause, group by and having clauses, select, where clause
**current_bigdatetime**

**Description**

Returns a bigintime value representing the current time with microsecond precision. The accuracy of the current time portion is limited by the accuracy of the system clock.

**Syntax**

`current_bigdatetime()`

**Parameters**

None.

**Examples**

**Example 1** Find the current bigdatetime:

```sql
select current_bigdatetime()
```

```
-------------------------------
Nov 25 1995 10:32:00.010101AM
```

**Example 2** Find the current bigdatetime:

```sql
select datepart(us, current_bigdatetime())
```

```
-------------------------------
010101
```

**Usage**

Finds the current date as it exists on the server.

**Standards**

ANSI SQL – Compliance level: Entry-level compliant.

**Permissions**

Any user can execute `current_date`.

**See also**

- Datatypes Date and time datatypes
- Commands `select`, `where` clause
- Functions `dateadd`, `datediff`, `datepart`, `datename`, `current_bigtime`
current_bigtime

Description
Returns a bigtime value representing the current time with microsecond precision. The accuracy of the current time portion is limited by the accuracy of the system clock.

Syntax
current_bigtime()

Parameters
None.

Examples
Example 1 Finds the current bigtime:

```sql
select current_bigtime()
```

```
10:32:00.010101AM
```

Example 2 Finds the current bigtime:

```sql
select datepart(us, current_bigtime())
```

```
01010
```

Usage
Finds the current date as it exists on the server.

Standards
ANSI SQL – Compliance level: Entry-level compliant.

Permissions
Any user can execute current_date.

See also
Datatypes Date and time datatypes

Commands select, where clause

Functions dateadd, datediff, datepart, datename, current_bigdatetime
**current_date**

Description          Returns the current date.

Syntax               `current_date()`

Parameters           None.

Examples

**Example 1** Identifies the current date with `datename`:

```
1> select datename(month, current_date())
2> go
-------------------------------
August
```

**Example 2** Identifies the current date with `datepart`:

```
1> select datepart(month, current_date())
2> go
-----------
8
```

(1 row affected)

Usage                  Finds the current date as it exists on the server.

Standards             ANSI SQL – Compliance level: Entry-level compliant.

Permissions           Any user can execute `current_date`.

See also

**Datatypes**  Date and time datatypes

**Commands**  `select`, `where clause`

**Functions**  `dateadd`, `datename`, `datepart`, `getdate`
current_time

Description
Returns the current time.

Syntax
current_time()

Parameters
None.

Examples
Example 1 Finds the current time:

```sql
1> select current_date()
2> go
--------------------
Aug 29 2003
(1 row affected)
```

Example 2 Use with datename:

```sql
1> select datename(minute, current_time())
2> go
--------------------------
45
(1 row affected)
```

Usage
Finds the current time as it exists on the server

Standards
ANSI SQL – Compliance level: Entry-level compliant.

Permissions
Any user can execute current_time.

See also
Datatypes Date and time datatypes
Commands select, where clause
Functions dateadd, datename, datepart, getdate
curunreservedpgs

Description
Returns the number of free pages in the specified disk piece.

Syntax
curunreservedpgs (dbid, lstart, unreservedpgs)

Parameters
- dbid
  is the ID for a database. These are stored in the db_id column of sysdatabases.
- lstart
  is a page within the disk piece for which pages are to be returned.
- unreservedpgs
  is the default value to return if the dbtable is presently unavailable for the requested database.

Examples

**Example 1** Returns the database name, device name, and the number of unreserved pages for each device fragment
If a database is open, curunreservedpgs takes the value from memory. If it is not in use, the value is taken from the third parameter you specify in curunreservedpgs. In this example, the value comes from the unreservedpgs column in the sysusages table.

```sql
select (dbid), d.name, curunreservedpgs(dbid, lstart, unreservedpgs) from sysusages u, sysdevices d where u.vdevno=d.vdevno and d.status &2 = 2
```

<table>
<thead>
<tr>
<th>name</th>
<th>dbid</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>master</td>
<td></td>
<td>1634</td>
</tr>
<tr>
<td>tempdb</td>
<td></td>
<td>423</td>
</tr>
<tr>
<td>model</td>
<td></td>
<td>423</td>
</tr>
<tr>
<td>pubs2</td>
<td></td>
<td>72</td>
</tr>
<tr>
<td>sybsystemdb</td>
<td></td>
<td>399</td>
</tr>
<tr>
<td>sybsystemprocs</td>
<td></td>
<td>6577</td>
</tr>
<tr>
<td>sybsyntax</td>
<td></td>
<td>359</td>
</tr>
</tbody>
</table>

(7 rows affected)

**Example 2** Displays the number of free pages on the segment for dbid starting on sysusages.lstart:

```sql
select curunreservedpgs (dbid, sysusages.lstart, 0)
```
curunreservedpgs, a system function, returns the number of free pages in a disk piece. For general information about system functions, see “System functions” on page 77.

If a database is open, the value returned by curunreservedpgs is taken from memory. If it is not in use, the value is taken from the third parameter you specify in curunreservedpgs.

ANSI SQL – Compliance level: Transact-SQL extension.

Any user can execute curunreservedpgs.

Functions  db_id, lct_admin
**data_pages**

### Description

Returns the number of pages used by the specified table, index, or a specific partition. The result does not include pages used for internal structures.

This function replaces `data_pgs` and `ptn_data_pgs` from versions of Adaptive Server earlier than 15.0.

### Syntax

```
data_pages(dbid, object_id [, indid [, ptnid]])
```

### Parameters

- **dbid**
  
is the database ID of the database that contains the data pages.

- **object_id**
  
is an object ID for a table, view, or other database object. These are stored in the `id` column of `sysobjects`.

- **indid**
  
is the index ID of the target index.

- **ptnid**
  
is the partition ID of the target partition.

### Examples

**Example 1**

Returns the number of pages used by the object with an object ID of 31000114 in the specified database (including any indexes):

```
select data_pages(5, 31000114)
```

**Example 2**

Returns the number of pages used by the object in the data layer, regardless of whether or not a clustered index exists:

```
select data_pages(5, 31000114, 0)
```

**Example 3**

Returns the number of pages used by the object in the index layer for a clustered index. This does not include the pages used by the data layer:

```
select data_pages(5, 31000114, 1)
```

**Example 4**

Returns the number of pages used by the object in the data layer of the specific partition, which in this case is 2323242432:

```
select data_pages(5, 31000114, 0, 2323242432)
```

### Usage

In the case of an APL (all-pages lock) table, if a clustered index exists on the table, then passing in an `indid` of:

- 0 – reports the data pages.
- 1 – reports the index pages.

All erroneous conditions return a value of zero, such as when the `object_id` does not exist in the current database, or the targeted `indid` or `ptnid` cannot be found.
Standards  
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions  
Any user can execute data_pages.

See also  
Functions  object_id, row_count
System procedure  sp_spaceused
datachange

Description
Measures the amount of change in the data distribution since update statistics last ran. Specifically, it measures the number of inserts, updates, and deletes that have occurred on the given object, partition, or column, and helps you determine if invoking update statistics would benefit the query plan.

Syntax
datachange(object_name, partition_name, column_name)

Parameters

object_name
is the object name in the current database.

partition_name
is the data partition name. This value can be null.

column_name
is the column name for which the datachange is requested. This value can be null.

Examples

Example 1 Provides the percentage change in the au_id column in the author_ptn partition:

select datachange("authors", "author_ptn", "au_id")

Example 2 Provides the percentage change in the authors table on the au_ptn partition. The null value for the column_name parameter indicates that this checks all columns that have histogram statistics and obtains the maximum datachange value from among them.

select datachange("authors", "au_ptn", null)

Usage

• The datachange function requires all three parameters.

• datachange is a measure of the inserts, deletes and updates but it does not count them individually. datachange counts an update as a delete and an insert, so each update contributes a count of 2 towards the datachange counter.

• The datachange built-in returns the datachange count as a percent of the number of rows, but it bases this percentage on the number of rows remaining, not the original number of rows. For example, if a table has five rows and one row is deleted, datachange reports a value of 25% since the current row count is 4 and the datachange counter is 1.

• datachange is expressed as a percentage of the total number of rows in the table, or partition if you specify a partition. The percentage value can be greater than 100 percent because the number of changes to an object can be much greater than the number of rows in the table, particularly when the number of deletes and updates happening to a table is very high.
• The value that datachange displays is the in-memory value. This can differ from the on-disk value because the on-disk value gets updated by the housekeeper, when you run sp_flushstats, or when an object descriptor gets flushed.

• The datachange values is not reset when histograms are created for global indexes on partitioned tables.

datachange is reset or initialized to zero when:

• New columns are added, and their datachange value is initialized.

• New partitions are added, and their datachange value is initialized.

• Data-partition-specific histograms are created, deleted or updated. When this occurs, the datachange value of the histograms is reset for the corresponding column and partition.

• Data is truncated for a table or partition, and its datachange value is reset.

• A table is repartitioned either directly or indirectly as a result of some other command, and the datachange value is reset for all the table’s partitions and columns.

• A table is unpartitioned, and the datachange value is reset for all columns for the table.

datachange has the following restrictions:

• datachange statistics are not maintained on tables in system tempdbs, user-defined tempdbs, system tables, or proxy tables.

• datachange updates are non-transactional. If you roll back a transaction, the datachange values are not rolled back, and these values can become inaccurate.

• If memory allocation for column-level counters fails, Adaptive Server tracks partition-level datachange values instead of column-level values.

• If Adaptive Server does not maintain column-level datachange values, it then resets the partition-level datachange values whenever the datachange values for a column are reset.

Permissions

Any user can execute datachange.
**datalength**

**Description**
Returns the actual length, in bytes, of the specified column or string.

**Syntax**
```
datalength(expression)
```

**Parameters**
- `expression` is a column name, variable, constant expression, or a combination of any of these that evaluates to a single value. `expression` can be of any datatype, and is usually a column name. If `expression` is a character constant, it must be enclosed in quotes.

**Examples**
Finds the length of the `pub_name` column in the `publishers` table:
```
select Length = datalength(pub_name)
from publishers
```
```
Length
-------
13
16
20
```

**Usage**
- `datalength`, a system function, returns the length of `expression` in bytes.
- For columns defined for the Unicode datatype, `datalength` returns the actual number of bytes of the data stored in each row. For example, this is what is returned if a unitext column `ut` contains row value `U+0041U+0042U+d800dc00`:
```
select datalength(ut) from unitable
```
```
-------------
8
```
- `datalength` finds the actual length of the data stored in each row. `datalength` is useful on `varchar`, `univarchar`, `varbinary`, `text`, and `image` datatypes, since these datatypes can store variable lengths (and do not store trailing blanks). When a `char` or `unichar` value is declared to allow nulls, Adaptive Server stores it internally as `varchar` or `univarchar`. For all other datatypes, `datalength` reports the defined length.
- `datalength` of any NULL data returns NULL.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute `datalength`.

**See also**
- **Functions** `char_length`, `col_length`
**dateadd**

**Description**
Adds an interval to a specified date or time. It takes three arguments: the datepart, a number, and a chronological expression. The datatype of the result is the same as the datatype of the last argument with a value equal to the original value plus the number of date parts. If the last argument is a bigtime, and the datepart is a year, month, or day, the result is the original bigtime argument.

**Syntax**
dateadd(date_part, integer, {date | time | bigtime | datetime, | bigdatetime})

**Parameters**
date_part
is a date part or abbreviation. For a list of the date parts and abbreviations recognized by Adaptive Server, see “Date parts” on page 73.

numeric
is an integer expression.

date expression
is an expression of type datetime, smalldatetime, bigdatetime, bigtime, date, time, or a character string in a datetime format.

**Examples**

**Example 1** Adds one million microseconds to a bigtime:

```sql
declare @a bigtime
select @a = "14:20:00.010101"
select dateadd(us, 1000000, @a)
--------------------------
2:20:01.010101PM
```

**Example 2** Adds 25 hours to a bigdatetime and the day will increment:

```sql
declare @a bigdatetime
select @a = "apr 12, 0001 14:20:00 "
select dateadd(hh, 25, @a)
--------------------------
Apr 13 0001   2:20PM
```

**Example 3** Displays the new publication dates when the publication dates of all the books in the titles table slip by 21 days:

```sql
select newpubdate = dateadd(day, 21, pubdate)
from titles
```

**Example 4** Adds one day to a date:

```sql
declare @a date
select @a = "apr 12, 9999"
select dateadd(dd, 1, @a)
--------------------------
```
Example 5  Subtracts five minutes to a time:

```sql
select dateadd(mi, -5, convert(time, "14:20:00"))
```

---

2:15PM

Example 6  Adds one day to a time and the time remains the same:

```sql
declare @a time
select @a = "14:20:00"
select dateadd(dd, 1, @a)
```

---

2:20PM

Example 7  Adds higher values resulting in the values rolling over to the next significant field, even though there are limits for each date_part, as with datetime values:

```sql
--Add 24 hours to a datetime
select dateadd(hh, 24, "4/1/1979")
```

---

Apr 2 1979 12:00AM

```sql
--Add 24 hours to a date
select dateadd(hh, 24, "4/1/1979")
```

---

Apr 2 1979

Usage

- `dateadd`, a date function, adds an interval to a specified date. For more information about date functions, see “Date functions” on page 73.

- `dateadd` takes three arguments: the date part, a number, and a date. The result is a datetime value equal to the date plus the number of date parts.

  - If the date argument is a smalldatetime value, the result is also a smalldatetime. You can use `dateadd` to add seconds or milliseconds to a smalldatetime, but such an addition is meaningful only if the result date returned by `dateadd` changes by at least one minute.

  - If a string is given as an argument in place of the chronological value the server interprets it as a datetime value regardless of its apparent precision. This default behavior may be changed by setting the configuration parameter `builtin date strings` or the set option `builtin_date_strings`. When these options are set the server will interpret strings given to chronological builtins as bigdatetimes. See the *System Administration Guide* for more information.
• When a datepart of microseconds is given to this built-in string values will always be interpreted as `bigdatetime`.

• Use the `datetime` datatype only for dates after January 1, 1753. `datetime` values must be enclosed in single or double quotes. Use the `date` datatype for dates from January 1, 0001 to 9999. `date` must be enclosed in single or double quotes. Use `char`, `nchar`, `varchar`, or `nvarchar` for earlier dates. Adaptive Server recognizes a wide variety of date formats. For more information, see “User-defined datatypes” on page 44 and “Datatype conversion functions” on page 63.

Adaptive Server automatically converts between character and `datetime` values when necessary (for example, when you compare a character value to a `datetime` value).

• Using the date part `weekday` or `dw` with `dateadd` is not logical, and produces spurious results. Use `day` or `dd` instead.

<table>
<thead>
<tr>
<th>Table 2-8: <code>date_part</code> recognized abbreviations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Date part</strong></td>
</tr>
<tr>
<td>Year</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Quarter</td>
</tr>
<tr>
<td>Month</td>
</tr>
<tr>
<td>Week</td>
</tr>
<tr>
<td>Day</td>
</tr>
<tr>
<td>dayofyear</td>
</tr>
<tr>
<td>Weekday</td>
</tr>
<tr>
<td>Hour</td>
</tr>
<tr>
<td>Minute</td>
</tr>
<tr>
<td>Second</td>
</tr>
<tr>
<td>millisecond</td>
</tr>
<tr>
<td>microsecond</td>
</tr>
</tbody>
</table>

Standards: ANSI SQL – Compliance level: Transact-SQL extension.

Permissions: Any user can execute `dateadd`.

See also: Date and time datatypes

Commands: `select`, `where clause`

Functions: `datediff`, `datename`, `datepart`, `getdate`
datediff

**Description**
Calculates the number of date parts between two specified dates or times. It takes three arguments. The first is a date part. The second and third are chronological values. For dates, times, datetimes and bigdatetimes, the result is a signed integer value equal to date2 and date1, in date parts.

• If the second or third argument is a date, and the date part is an hour, minute, second, millisecond, or microsecond, the dates are treated as midnight.

• If the second or third argument is a time, and the date part is a year, month, or day, then zero is returned.

• `datediff` results are truncated, not rounded when the result is not an even multiple of the date part.

• For the smaller time units, there are overflow values and the function returns an overflow error if you exceed these limits.

**Syntax**
```
datediff(datepart, {date, date | time, time | bigtime, bigtime | datetime, datetime | bigdatetime, bigdatetime})
```

**Parameters**
- `datepart` is a date part or abbreviation. For a list of the date parts and abbreviations recognized by Adaptive Server, see “Date parts” on page 73.
- `date expression1` is an expression of type datetime, smalldatetime, bigdatetime, bigtime, date, time, or a character string in a datetime format.
- `date expression2` is an expression of type datetime, smalldatetime, bigdatetime, bigtime, date, time, or a character string in a datetime format.

**Examples**

**Example 1** Returns the number of microseconds between two bigdatetimes:
```
declare @a bigdatetime
declare @b bigdatetime
select @a = "apr 1, 1999 00:00:00.000000"
select @b = "apr 2, 1999 00:00:00.000000"
select datediff(us, @a, @b) --------------------------
86400000000
```

**Example 2** Returns the overflow size of milliseconds return value:
```
select datediff(ms, convert(bigdatetime, "4/1/1753"), convert(bigdatetime, "4/1/9999"))
Msg 535, Level 16, State 0:
Line 2:
```

Adaptive Server Enterprise
Datediff returns the number of date parts difference between two dates.

**Example 3** Finds the number of days that have elapsed between pubdate and the current date (obtained with the getdate function):

```sql
select newdate = datediff(day, pubdate, getdate())
from titles
```

**Example 4** Finds the number of hours between two times:

```sql
declare @a time
declare @b time
select @a = "20:43:22"
select @b = "10:43:22"
select datediff(hh, @a, @b)
------------
-10
```

**Example 5** Finds the number of hours between two dates:

```sql
declare @a date
declare @b date
select @a = "apr 1, 1999"
select @b = "apr 2, 1999"
select datediff(hh, @a, @b)
------------
24
```

**Example 6** Finds the number of days between two times:

```sql
declare @a time
declare @b time
select @a = "20:43:22"
select @b = "10:43:22"
select datediff(dd, @a, @b)
------------
0
```

**Example 7** Returns the overflow size of milliseconds return value:

```sql
select datediff(ms, convert(date, "4/1/1753"), convert(date, "4/1/9999"))
```

Usage

- `datediff`, a date function, calculates the number of date parts between two specified dates. For more information about date functions, see “Date functions” on page 73.
**datediff**

- `datediff` takes three arguments. The first is a date part. The second and third are dates. The result is a signed integer value equal to `date2 - date1`, in date parts.

- `datediff` produces results of datatype `int`, and causes errors if the result is greater than 2,147,483,647. For milliseconds, this is approximately 24 days, 20:31.846 hours. For seconds, this is 68 years, 19 days, 3:14:07 hours.

- `datediff` results are always truncated, not rounded, when the result is not an even multiple of the date part. For example, using `hour` as the date part, the difference between “4:00AM” and “5:50AM” is 1.

  When you use `day` as the date part, `datediff` counts the number of midnights between the two times specified. For example, the difference between January 1, 1992, 23:00 and January 2, 1992, 01:00 is 1; the difference between January 1, 1992 00:00 and January 1, 1992, 23:59 is 0.

- The `month` datepart counts the number of first-of-the-months between two dates. For example, the difference between January 25 and February 2 is 1; the difference between January 1 and January 31 is 0.

- When you use the date part `week` with `datediff`, you see the number of Sundays between the two dates, including the second date but not the first. For example, the number of weeks between Sunday, January 4 and Sunday, January 11 is 1.

- If you use `smalldatetime` values, they are converted to `datetime` values internally for the calculation. Seconds and milliseconds in `smalldatetime` values are automatically set to 0 for the purpose of the difference calculation.

- If the second or third argument is a date, and the `datepart` is `hour`, `minute`, `second`, or `millisecond`, the dates are treated as midnight.

- If the second or third argument is a time, and the `datepart` is `year`, `month`, or `day`, then 0 is returned.

- `datediff` results are truncated, not rounded, when the result is not an even multiple of the date part.
• If a string is given as an argument in place of the chronological value the server interprets it as a `datetime` value regardless of its apparent precision. This default behavior may be changed by setting the configuration parameter `builtin_date_strings` or the set option `builtin_date_strings`. When these options are set the server will interpret strings given to chronological builtins as `bigdatetime`. See the *System Administration Guide* for more information.

• When a datepart of microseconds is given to this builtin string values will always be interpreted as `bigdatetime`.

• For the smaller time units, there are overflow values, and the function returns an overflow error if you exceed these limits:
  - Microseconds: approx 3 days
  - Milliseconds: approx 24 days
  - Seconds: approx 68 years
  - Minutes: approx 4083 years
  - Others: No overflow limit

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute `datediff`.

**See also**
- Datatypes Date and time datatypes
- Commands `select`, `where clause`
- Functions `dateadd`, `datename`, `datepart`, `getdate`
datename

Description
Returns the specified datepart (the first argument) of the specified date or time (the second argument) as a character string. Takes a date, time, bigdatetime, bigtime, datetime, or smalldatetime value as its second argument.

Syntax
datename(datepart [date | time | bigtime | datetime | bigdatetime])

Parameters
datepart
is a date part or abbreviation. For a list of the date parts and abbreviations recognized by Adaptive Server, see “Date parts” on page 73.

date_expression
is an expression of type datetime, smalldatetime, bigdatetime, bigtime, time or a character string in a datetime format.

Examples
Example 1 Finds the month name of a bigdatetime:

```
declare @a bigdatetime
select @a = "apr 12, 0001 00:00:00.010101"
select datename(mm, @a)
------------------------------
April
```

Example 2 Assumes a current date of November 20, 2000:

```
select datename(month, getdate())
November
```

Example 3 Finds the month name of a date:

```
declare @a date
select @a = "apr 12, 0001"
select datename(mm, @a)
------------------------------
April
```

Example 4 Finds the seconds of a time:

```
declare @a time
select @a = "20:43:22"
select datename(ss, @a)
------------------------------
22
```

Usage
• datename, a date function, returns the name of the specified part (such as the month “June”) of a datetime or smalldatetime value, as a character string. If the result is numeric, such as “23” for the day, it is still returned as a character string.
• For more information about date functions, see “Date functions” on page 73.

• The date part weekday or dw returns the day of the week (Sunday, Monday, and so on) when used with datename.

• Since smalldatetime is accurate only to the minute, when a smalldatetime value is used with datename, seconds and milliseconds are always 0.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute datename.

See also
Datatypes  Date and time datatypes
Commands  select, where clause
Functions  dateadd, datename, datepart, getdate
datepart

**Description**

Returns the specified datepart in the first argument of the specified date, and the second argument, as an integer. Takes a `date`, `time`, `datetime`, `bigdatetime`, `bigtime`, or `smalldatetime` value as its second argument. If the datepart is `hour`, `minute`, `second`, `millisecond`, or `microsecond`, the result is 0.

**Syntax**

```
datepart(date_part (date | time | datetime | bigtime | bigdatetime))
```

**Parameters**

`date_part` is a date part. Table 2-9 lists the date parts, the abbreviations recognized by `datepart`, and the acceptable values.

<table>
<thead>
<tr>
<th>Date part</th>
<th>Abbreviation</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>year</td>
<td>yy</td>
<td>1753 – 9999 (2079 for smalldatetime). 0001 to 9999 for date</td>
</tr>
<tr>
<td>quarter</td>
<td>qq</td>
<td>1 – 4</td>
</tr>
<tr>
<td>month</td>
<td>mm</td>
<td>1 – 12</td>
</tr>
<tr>
<td>week</td>
<td>wk</td>
<td>1 – 54</td>
</tr>
<tr>
<td>day</td>
<td>dd</td>
<td>1 – 31</td>
</tr>
<tr>
<td>dayofyear</td>
<td>dy</td>
<td>1 – 366</td>
</tr>
<tr>
<td>weekday</td>
<td>dw</td>
<td>1 – 7 (Sun. – Sat.)</td>
</tr>
<tr>
<td>hour</td>
<td>hh</td>
<td>0 – 23</td>
</tr>
<tr>
<td>minute</td>
<td>mi</td>
<td>0 – 59</td>
</tr>
<tr>
<td>second</td>
<td>ss</td>
<td>0 – 59</td>
</tr>
<tr>
<td>millisecond</td>
<td>ms</td>
<td>0 – 999</td>
</tr>
<tr>
<td>microsecond</td>
<td>us</td>
<td>0 - 999999</td>
</tr>
<tr>
<td>calweekofyear</td>
<td>cwk</td>
<td>1 – 53</td>
</tr>
<tr>
<td>calyearofweek</td>
<td>cyr</td>
<td>1753 – 9999 (2079 for smalldatetime). 0001 to 9999 for date</td>
</tr>
<tr>
<td>caldayofweek</td>
<td>cdw</td>
<td>1 – 7</td>
</tr>
</tbody>
</table>

Table 2-9: Date parts and their values
When you enter a year as two digits (yy):

- Numbers less than 50 are interpreted as 20yy. For example, 01 is 2001, 32 is 2032, and 49 is 2049.

- Numbers equal to or greater than 50 are interpreted as 19yy. For example, 50 is 1950, 74 is 1974, and 99 is 1999.

For datetime, smalldatetime, and time types milliseconds can be preceded by either a colon or a period. If preceded by a colon, the number means thousandths of a second. If preceded by a period, a single digit means tenths of a second, two digits mean hundredths of a second, and three digits mean thousandths of a second. For example, “12:30:20:1” means twenty and one-thousandth of a second past 12:30; “12:30:20.1” means twenty and one-tenth of a second past 12:30.

Microseconds must be preceded by a decimal point and represent fractions of a second.

date_expression

is an expression of type datetime, smalldatetime, bigdatetime, bigtime, date, time, or a character string in a datetime format.

Examples

Example 1 Finds the microseconds of a bigdatetime:

declare @a bigdatetime
select @a = "apr 12, 0001 12:00:00.000001"
select datepart(us, @a)
-----------
000001

Example 2 Assumes a current date of November 25, 1995:

select datepart(month, getdate())
-----------
11

Example 3 Returns the year of publication from traditional cookbooks:

select datepart(year, pubdate) from titles
   where type = "trad_cook"
-----------
1990
1985
1987

Example 4
select datepart(cwk,'1993/01/01')
-----------
53

Example 5
select datepart(cyr,'1993/01/01')
-----------
1992

Example 6
select datepart(cdw,'1993/01/01')
-----------
5

Example 7 Find the hours in a time:

declare @a time
select @a = "20:43:22"
select datepart(hh, @a)
-----------
20

Example 8 Returns 0 (zero) if an hour, minute, or second portion is requested from a date using datename or datepart) the result is the default time; Returns the default date of Jan 1 1990 if month, day, or year is requested from a time using datename or datepart:

--Find the hours in a date
declare @a date
select @a = "apr 12, 0001"
select datepart(hh, @a)
-----------
0

--Find the month of a time
declare @a time
select @a = "20:43:22"
select datename(mm, @a)
----------------------
January

When you give a null value to a datetime function as a parameter, NULL is returned.

Usage

• datepart, a date function, returns an integer value for the specified part of a datetime value. For more information about date functions, see “Date functions” on page 73.
datepart returns a number that follows ISO standard 8601, which defines
the first day of the week and the first week of the year. Depending on
whether the datepart function includes a value for calweeekofyear,
calyearofweek, or caldayofweek, the date returned may be different for the
same unit of time. For example, if Adaptive Server is configured to use
U.S. English as the default language, the following returns 1988:

```
datepart(cyr, "1/1/1989")
```

However, the following returns 1989:

```
datepart(yy, "1/1/1989")
```

This disparity occurs because the ISO standard defines the first week of
the year as the first week that includes a Thursday and begins with
Monday.

For servers using U.S. English as their default language, the first day of the
week is Sunday, and the first week of the year is the week that contains
January 4th.

- The date part weekday or dw returns the corresponding number when used
  with datepart. The numbers that correspond to the names of weekdays
  depend on the datefirst setting. Some language defaults (including
  us_english) produce Sunday=1, Monday=2, and so on; others produce
  Monday=1, Tuesday=2, and so on. You can change the default behavior on
  a per-session basis with set datefirst. See the datefirst option of the set
  command for more information.

- calweeekofyear, which can be abbreviated as cwk, returns the ordinal
  position of the week within the year. calyearofweek, which can be
  abbreviated as Cyr, returns the year in which the week begins.
  caldayofweek, which can abbreviated as cdw, returns the ordinal position
  of the day within the week. You cannot use calweeekofyear, calyearofweek,
  and caldayofweek as date parts for dateadd, datediff, and datename.

- Since datetime and time are only accurate to 1/300th of a second, when
  these datatypes are used with datepart, milliseconds are rounded to the
  nearest 1/300th second.

- Since smalldatetime is accurate only to the minute, when a smalldatetime
  value is used with datepart, seconds and milliseconds are always 0.

- The values of the weekday date part are affected by the language setting.
See also

**Datatypes**  Date and time datatypes

**Commands**  select, where clause

**Functions**  dateadd, datediff, datename, getdate
**day**

**Description**
Returns an integer that represents the day in the datepart of a specified date.

**Syntax**
`day(date_expression)`

**Parameters**
`date_expression` is an expression of type `datetime`, `smalldatetime`, `date`, or a character string in a datetime format.

**Examples**
Returns the integer 02:
```
    day("11/02/03")
```

**Usage**
`day(date_expression)` is equivalent to `datepart(dd,date_expression)`.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute `day`.

**See also**
*Datatypes*  
datetime, smalldatetime, date, time

*Functions*  
datepart, month, year
db_attr

Description
Returns the durability, dml_logging, and template settings for the specified database.

Syntax
$db\_attr('database\_name' | database\_ID | NULL, 'attribute')$

Parameters
database\_name
name of the database.

database\_ID
ID of the database

NULL
if included, $db\_attr$ reports on the current database

attribute
is one of:

- help – display $db\_attr$ usage information.
- durability – returns durability of the given database. full, at\_shutdown, or no\_recovery.
- dml\_logging – returns the value for DML logging for specified database: full or minimal.
- template – returns the name of the template database used for the specified database. If no database was used as a template to create the database, returns NULL.

Examples

**Example 1** Returns the syntax for $db\_attr$:

```sql
select db_attr(0, "help")
Usage: db_attr('dbname' | dbid | NULL, 'attribute')
List of options in attributes table:
  0 : help
  1 : durability
  2 : dml\_logging
  3 : template

-----------------------------------------------------
NULL

select abs(-1)
-----------
1
```

**Example 2** Selects the name, durability setting, dml\_logging setting and template used from sysdatabases:
select name = convert(char(20), name),
durability = convert(char(15), db_attr(name, "durability")),
dml_logging = convert(char(15), db_attr(dbid, "dml_logging")),
template = convert(char(15), db_attr(dbid, "template"))
from sysdatabases

<table>
<thead>
<tr>
<th>name</th>
<th>durability</th>
<th>dml_logging</th>
<th>template</th>
</tr>
</thead>
<tbody>
<tr>
<td>master</td>
<td>full</td>
<td>full</td>
<td>NULL</td>
</tr>
<tr>
<td>model</td>
<td>full</td>
<td>full</td>
<td>NULL</td>
</tr>
<tr>
<td>tempdb</td>
<td>no_recovery</td>
<td>full</td>
<td>NULL</td>
</tr>
<tr>
<td>sybsystemdb</td>
<td>full</td>
<td>full</td>
<td>NULL</td>
</tr>
<tr>
<td>sybsystemprocs</td>
<td>full</td>
<td>full</td>
<td>NULL</td>
</tr>
<tr>
<td>repro</td>
<td>full</td>
<td>full</td>
<td>NULL</td>
</tr>
<tr>
<td>imdb</td>
<td>no_recovery</td>
<td>full</td>
<td>db1</td>
</tr>
<tr>
<td>db</td>
<td>full</td>
<td>full</td>
<td>NULL</td>
</tr>
<tr>
<td>at_shutdown_db</td>
<td>at_shutdown</td>
<td>full</td>
<td>NULL</td>
</tr>
<tr>
<td>db1</td>
<td>full</td>
<td>full</td>
<td>NULL</td>
</tr>
<tr>
<td>dml</td>
<td>at_shutdown</td>
<td>minimal</td>
<td>NULL</td>
</tr>
</tbody>
</table>

Example 3 Runs db_attr against the DoesNotExist database, which does not exist:

```sql
select db_attr("DoesNotExist", "durability")
```

NULL

Example 4 Runs db_attr against a database with an ID of 12345, which does not exist:

```sql
select db_attr(12345, "durability")
```

NULL

Example 5 Runs db_attr against an attribute that does not exist:

```sql
select db_attr(1, "Cmd Does Not Exist")
```

NULL

Usage

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute db_attr.

See also

Functions
db_id

Description
Returns the ID number of the specified database.

Syntax
db_id(database_name)

Parameters
database_name
is the name of a database. database_name must be a character expression. If it is a constant expression, it must be enclosed in quotes.

Examples
Returns the ID number of sybsystemprocs:

    select db_id("sybsystemprocs")
    -------
    4

Usage
• db_id, a system function, returns the database ID number.

• If you do not specify a database_name, db_id returns the ID number of the current database.

• For general information about system functions, see “System functions” on page 77.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute db_id.

See also
Functions db_name, object_id
db_instanceid

Description
Cluster environments only – returns the ID of the owning instance of a specified local temporary database. Returns NULL if the specified database is a global temporary database or a nontemporary database.

Syntax

\[
\text{db_instanceid}(\text{database}_\text{id}) \\
\text{db_instanceid}(\text{database}_\text{name})
\]

Parameters

\begin{itemize}
  \item \text{database}_\text{id} \quad \text{ID of the database.}
  \item \text{database}_\text{name} \quad \text{name of the database}
\end{itemize}

Examples

Returns the owning instance for database ID 5

\[
\text{select db_instanceid(5)}
\]

Usage

\begin{itemize}
  \item Access to a local temporary database is allowed only from the owning instance. \text{db_instanceid} determines whether the specified database is a local temporary database, and the owning instance for the local temporary database. You can then connect to the owning instance and access its local temporary database.
  \item You must include a parameter with \text{db_instanceid}.
\end{itemize}

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can run \text{sdc_intempdbconfig}.
**db_name**

**Description**
Returns the name of the database where the ID number is specified.

**Syntax**
```
db_name([database_id])
```

**Parameters**
- `database_id`, a numeric expression for the database ID (stored in `sysdatabases.dbid`).

**Examples**
- **Example 1** Returns the name of the current database:
  ```
  select db_name()
  ```

- **Example 2** Returns the name of database ID 4:
  ```
  select db_name(4)
  ```

**Usage**
- If no `database_id` is supplied, `db_name` returns the name of the current database.
- For general information about system functions, see “System functions” on page 77.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute `db_name`.

**See also**
- **Functions** `col_name`, `db_id`, `object_name`
db_recovery_status

Description
Cluster environments only – returns the recovery status of the specified database. Returns the recovery status of the current database if you do not include a value for database_ID or database_name.

Syntax
`db_recovery_status((database_ID | database_name))`

Parameters
- `database_ID` is the ID of the database whose recovery status you are requesting.
- `database_name` is the name of the database whose recovery status you are requesting.

Examples
**Example 1** Returns the recovery status of the current database:
```
select db_recovery_status()
```

**Example 2** Returns the recovery status of the database with named test:
```
select db_recovery_status("test")
```

**Example 3** Returns the recovery status of a database with a database id of 8:
```
select db_recovery_status(8)
```

Usage
A return value of 0 indicates the database is not in node-failover recovery. A return value of 1 indicates the database is in node-failover recovery.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute `db_recovery_status`.
**degrees**

Description

Returns the size, in degrees, of an angle with the specified number of radians.

Syntax

```
degrees(numeric)
```

Parameters

- `numeric` is a number, in radians, to convert to degrees.

Examples

```
select degrees(45)
```

```
-------------
 2578
```

Usage

- `degrees`, a mathematical function, converts radians to degrees. Results are of the same type as the numeric expression.

  For numeric and decimal expressions, the results have an internal precision of 77 and a scale equal to that of the expression.

- For general information about mathematical functions, see “Mathematical functions” on page 74.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute `degrees`.

See also

Function `radians`


**derived_stat**

**Description**  
Returns derived statistics for the specified object and index.

**Syntax**  
```
derived_stat("object_name" | object_id, 
    index_name | index_id, 
    ["partition_name" | partition_id], 
    "statistic")
```

**Parameters**

- **object_name**  
  is the name of the object you are interested in. If you do not specify a fully qualified object name, derived_stat searches the current database.

- **object_id**  
  is an alternative to **object_name**, and is the object ID of the object you are interested in. **object_id** must be in the current database.

- **index_name**  
  is the name of the index, belonging to the specified object that you are interested in.

- **index_id**  
  is an alternative to **index_name**, and is the index ID of the specified object that you are interested in.

- **partition_name**  
  is the name of the partition, belonging to the specific partition that you are interested in. **partition_name** is optional. When you use **partition_name** or **partition_id**, Adaptive Server returns statistics for the target partition, instead of for the entire object.

- **partition_id**  
  is an alternative to **partition_name**, and is the partition ID of the specified object that you are interested in. **partition_id** is optional.
"statistic"

the derived statistic to be returned. Available statistics are:

- data page cluster ratio or dpcr – the data page cluster ratio for the object/index pair
- index page cluster ratio or ipcr – the index page cluster ratio for the object/index pair
- data row cluster ratio or drcr – the data row cluster ratio for the object/index pair
- large io efficiency or lgio – the large I/O efficiency for the object/index pair
- space utilization or sput – the space utilization for the object/index pair

Examples

Example 1 Selects the space utilization for the titleidind index of the titles table:

```sql
select derived_stat("titles", "titleidind", "space utilization")
```

Example 2 Selects the data page cluster ratio for index ID 2 of the titles table. Note that you can use either "dpcr" or "data page cluster ratio":

```sql
select derived_stat("titles", 2, "dpcr")
```

Example 3 Statistics are reported for the entire object, as neither the partition ID nor name is not specified:

1> select derived_stat(object_id("t1"), 2, "drcr")
2> go

```
---------------------------
0.576923
```

Example 4 Reports the statistic for the partition tl_928003396:

1> select derived_stat(object_id("t1"), 0, "tl_928003306", "drcr")
2> go

```
---------------------------
1.000000
```

(1 row affected)

Example 5 Selects derived statistics for all indexes of a given table, using data from syspartitions:

```sql
select convert(varchar(30), name) as name, indid,
       convert(decimal(5, 3), derived_stat(id, indid, 'sput')) as 'sput',
       convert(decimal(5, 3), derived_stat(id, indid, 'dpcr')) as 'dpcr',
       convert(decimal(5, 3), derived_stat(id, indid, 'drcr')) as 'drcr',
```
convert(decimal(5, 3), derived_stat(id, indid, 'lgio')) as 'lgio' 
from syspartitions where id = object_id('titles')
go

titleendid_2133579608  1  0.895  1.000  1.000  1.000  
titleendid_2133579608  2  0.000  1.000  0.688  1.000

(2 rows affected)

Example 6  Selects derived statistics for all indexes and partitions of a
partitioned table. Here, mymsgs_r4 is a roundrobin partitioned table that is
created with a global index and a local index.

1> select * into mymsgs_r4 partition by roundrobin 4 lock datarows
2> from master..sysmessages
2> go
(7597 rows affected)
1> create clustered index mymsgs_r4_clustind on mymsgs_r4(error, severity)
2> go
1> create index mymsgs_r4_ncind1 on mymsgs_r4(severity)
2> go
1> create index mymsgs_r4_ncind2 on mymsgs_r4(langid, dlevel) local index
2> go
2> update statistics mymsgs_r4
1>
2> select convert(varchar(10), object_name(id)) as name,
3> (select convert(varchar(20), i.name) from sysindexes i
4>   where i.id = p.id and i.indid = p.indid),
5> convert(varchar(30), name) as ptnname, indid,
6> convert(decimal(5, 3), derived_stat(id, indid, partitionid, 'sput')) as 'sput',
7> convert(decimal(5, 3), derived_stat(id, indid, partitionid, 'dpcr')) as 'dpcr',
8> convert(decimal(5, 3), derived_stat(id, indid, partitionid, 'drcr')) as 'drcr',
9> convert(decimal(5, 3), derived_stat(id, indid, partitionid, 'lgio')) as 'lgio'
10> from syspartitions p
11> where id = object_id('mymsgs_r4')

<table>
<thead>
<tr>
<th>name</th>
<th>ptnname</th>
<th>indid</th>
<th>sput</th>
<th>dpcr</th>
<th>drcr</th>
<th>lgio</th>
</tr>
</thead>
<tbody>
<tr>
<td>mymsgs_r4</td>
<td>mymsgs_r4</td>
<td>0</td>
<td>0.90</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>mymsgs_r4</td>
<td>mymsgs_r4</td>
<td>0</td>
<td>0.90</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>mymsgs_r4</td>
<td>mymsgs_r4</td>
<td>0</td>
<td>0.89</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>mymsgs_r4</td>
<td>mymsgs_r4</td>
<td>0</td>
<td>0.90</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>mymsgs_r4</td>
<td>mymsgs_r4_clustind</td>
<td>2</td>
<td>0.83</td>
<td>0.995</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>mymsgs_r4</td>
<td>mymsgs_r4_ncind1</td>
<td>3</td>
<td>0.99</td>
<td>0.445</td>
<td>0.88</td>
<td>1.000</td>
</tr>
<tr>
<td>mymsgs_r4</td>
<td>mymsgs_r4_ncind2</td>
<td>4</td>
<td>0.15</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>mymsgs_r4</td>
<td>mymsgs_r4_ncind2</td>
<td>4</td>
<td>0.88</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>mymsgs_r4</td>
<td>mymsgs_r4_ncind2</td>
<td>4</td>
<td>0.877</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
</tr>
</tbody>
</table>
Example 7 Selects derived statistics for all allpages-locked tables in the
current database:

```sql
2> select convert(varchar(10), object_name(id)) as name
3> (select convert(varchar(20), i.name) from sysindexes i
4>   where i.id = p.id and i.indid = p.indid),
5> convert(varchar(30), name) as ptnname, indid,
6> convert(decimal(5, 3), derived_stat(id, indid, partitionid, 'sput')) as 'sput',
7> convert(decimal(5, 3), derived_stat(id, indid, partitionid, 'dpcr')) as 'dpcr',
8> convert(decimal(5, 3), derived_stat(id, indid, partitionid, 'drcr')) as 'drcr',
9> convert(decimal(5, 3), derived_stat(id, indid, partitionid, 'lgio')) as 'lgio'
10> from syspartitions p
11> where lockscheme(id) = "allpages"
12> and (select o.type from sysobjects o where o.id = p.id) = 'U'
```

<table>
<thead>
<tr>
<th>name</th>
<th>ptnname</th>
<th>indid</th>
<th>sput</th>
<th>dpcr</th>
<th>drcr</th>
<th>lgio</th>
</tr>
</thead>
<tbody>
<tr>
<td>stores</td>
<td>stores</td>
<td>0.276</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>discounts</td>
<td>discounts</td>
<td>0.075</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>au_pix</td>
<td>au_pix</td>
<td>0.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>au_pix</td>
<td>tau_pix</td>
<td>255</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>blurs</td>
<td>blurs</td>
<td>0.055</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>blurs</td>
<td>tblurbs</td>
<td>255</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>t1apl</td>
<td>t1apl</td>
<td>0.095</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>t1apl</td>
<td>t1apl_ncind</td>
<td>0.095</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>t1apl</td>
<td>t1apl_ncind_local</td>
<td>2.149</td>
<td>0.000</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>t1apl</td>
<td>t1apl_ncind_local</td>
<td>3.066</td>
<td>0.000</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>authors</td>
<td>auidind</td>
<td>1.066</td>
<td>0.000</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>authors</td>
<td>aumind</td>
<td>2.030</td>
<td>0.000</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>publishers</td>
<td>pubind</td>
<td>1.059</td>
<td>0.000</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>roysched</td>
<td>roysched</td>
<td>0.324</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>roysched</td>
<td>titleidind</td>
<td>2.777</td>
<td>1.000</td>
<td>0.941</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>sales</td>
<td>salesind</td>
<td>1.444</td>
<td>0.000</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>salesdetail</td>
<td>salesdetail</td>
<td>0.614</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>salesdetail</td>
<td>titleidind</td>
<td>0.518</td>
<td>1.000</td>
<td>0.752</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>salesdetail</td>
<td>salesdetailind</td>
<td>0.794</td>
<td>1.000</td>
<td>0.726</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>titleautho</td>
<td>taind</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>titleautho</td>
<td>auidind</td>
<td>2.825</td>
<td>0.000</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>titles</td>
<td>titleidind</td>
<td>3.223</td>
<td>0.000</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>titles</td>
<td>titleind</td>
<td>1.085</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
</tr>
</tbody>
</table>

(27 rows affected)
Usage
- `derived_stat` returns a double precision value.
- The values returned by `derived_stat` match the values presented by the `optdiag` utility.
- If the specified object or index does not exist, `derived_stat` returns NULL.
- Specifying an invalid statistic type results in an error message.
- Using the optional `partition_name` or `partition_id` reports the requested statistic for the target partition; otherwise, `derived_stat` reports the statistic for the entire object.
- If you provide:
  - Four arguments – `derived_stat` uses the third argument as the partition, and returns derived statistics on the fourth argument.
  - Three arguments – `derived_stat` assumes you did not specify a partition, and returns derived statistic specified by the third argument.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Only the table owner can execute `derived_stat`.

See also
Document: *Performance and Tuning Guide* for:
- “Access Methods and Query Costing for Single Tables”
- “Statistics Tables and Displaying Statistics with `optdiag`”

Utility `optdiag`
difference

Description	Returns the difference between two soundex values.

Syntax	difference(expr1, expr2)

Parameters

expr1
is a character-type column name, variable, or constant expression of char, varchar, nchar, nvarchar, or unichar type.

expr2
is another character-type column name, variable, or constant expression of char, varchar, nchar, nvarchar, or unichar type.

Examples

Example 1

select difference("smithers", "smothers")
---------
4

Example 2

select difference("smothers", "brothers")
---------
2

Usage

• difference, a string function, returns an integer representing the difference between two soundex values.

• The difference function compares two strings and evaluates the similarity between them, returning a value from 0 to 4. The best match is 4.

  The string values must be composed of a contiguous sequence of valid single- or double-byte roman letters.

• If expr1 or expr2 is NULL, returns NULL.

• If you give a varchar expression is given as one parameter and a unichar expression as the other, the varchar expression is implicitly converted to unichar (with possible truncation).

• For general information about string functions, see “String functions” on page 76.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute difference.

See also

Function soundex
exp

Description
Returns the value that results from raising the constant to the specified power.

Syntax
exp(approx_numeric)

Parameters
approx_numeric
is any approximate numeric (float, real, or double precision) column name, variable, or constant expression.

Examples
select exp(3)
-------------------
20.085537

Usage
• exp, a mathematical function, returns the exponential value of the specified value.
• For general information about mathematical functions, see “Mathematical functions” on page 74.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute exp.

See also
Functions log, log10, power
**floor**

Description

Returns the largest integer that is less than or equal to the specified value.

Syntax

`floor(numeric)`

Parameters

`numeric`

is any exact numeric (`numeric`, `dec`, `decimal`, `tinyint`, `smallint`, `int`, or `bigint`), approximate numeric (`float`, `real`, or `double precision`), or money column, variable, constant expression, or a combination of these.

Examples

**Example 1**

```
select floor(123)
-----------
  123
```

**Example 2**

```
select floor(123.45)
-------
  123
```

**Example 3**

```
select floor(1.2345E2)
---------------------
  123.000000
```

**Example 4**

```
select floor(-123.45)
-------
  -124
```

**Example 5**

```
select floor(-1.2345E2)
----------------------
  -124.000000
```

**Example 6**

```
select floor($123.45)
---------------------
  123.00
```
Usage

- floor, a mathematical function, returns the largest integer that is less than or equal to the specified value. Results are of the same type as the numeric expression.

  For numeric and decimal expressions, the results have a precision equal to that of the expression and a scale of 0.

- For general information about mathematical functions, see “Mathematical functions” on page 74.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute floor.

See also

Functions abs, ceiling, round, sign
get_appcontext

Description
Returns the value of the attribute in a specified context. get_appcontext is a built-in function provided by the Application Context Facility (ACF).

Syntax
get_appcontext("context_name", "attribute_name")

Parameters
- context_name is a row specifying an application context name, saved as datatype char(30).
- attribute_name is a row specifying an application context attribute name, saved as char(30).

Examples
Example 1 Shows VALUE1 returned for ATTR1.

```sql
select get_appcontext("CONTEXT1", "ATTR1")
------------
VALUE1
```

ATTR1 does not exist in CONTEXT2:

```sql
select get_appcontext("CONTEXT2", "ATTR1")
```

Example 2 Shows the result when a user without appropriate permissions attempts to get the application context.

```sql
select get_appcontext("CONTEXT1", "ATTR2", "VALUE1")
```

Select permission denied on built-in get_appcontext, database dbid

-1

Usage
- This function returns 0 for success and -1 for failure.
- If the attribute you require does not exist in the application context, get_appcontext returns NULL.
- get_appcontext saves attributes as char datatypes. If you are creating an access rule that compares the attribute value to other datatypes, the rule should convert the char data to the appropriate datatype.
- All arguments for this function are required.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Permissions depend on the user profile and the application profile, and are stored by the ACF.

See also
For more information on the ACF, see “Row-level access control” in Chapter 11, “Managing User Permissions” of the System Administration Guide.

Functions
get_appcontext, list_appcontext, rm_appcontext, set_appcontext
getdate

Description
Returns the current system date and time.

Syntax
getdate()

Parameters
None.

Examples
Example 1 Assumes a current date of November 25, 1995, 10:32 a.m.:

```sql
select getdate()
```

Nov 25 1995 10:32AM

Example 2 Assumes a current date of November:

```sql
select datepart(month, getdate())
```

11

Example 3 Assumes a current date of November:

```sql
select datename(month, getdate())
```

November

Usage
- `getdate`, a date function, returns the current system date and time.
- For more information about date functions, see “Date functions” on page 73.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute getdate.

See also
Datatypes Date and time datatypes
Functions dateadd, datediff, datename, datepart
**getutcdate**

**Description**
Returns a date and time where the value is in Universal Coordinated Time (UTC). getutcdate is calculated each time a row is inserted or selected.

**Syntax**
getutcdate()

**Examples**
```
insert t1 (c1, c2, c3) select c1, getutcdate(),
getdate() from t2
```

**See also**
Functions  biginttohex, convert
has_role

Description
Returns information about whether the user has been granted the specified role.

Syntax
has_role ("role_name", option)

Parameters
role_name
is the name of a system or user-defined role.

option
allows you to limit the scope of the information returned. Currently, the only option supported is 1, which suppresses auditing.

Examples
Example 1 Creates a procedure to check if the user is a System Administrator:

create procedure sa_check as
begin
    if (has_role("sa_role", 0) > 0)
        begin
            print "You are a System Administrator."
            return(1)
        end
end

Example 2 Checks that the user has been granted the System Security Officer role:

select has_role("sso_role", 1)

Example 3 Checks that the user has been granted the Operator role:

select has_role("oper_role", 1)

Usage
• has_role functions the same way proc_role does. Beginning with Adaptive Server version 15.0, Sybase supports—and recommends—that you use has_role instead of proc_role. You need not, however, convert all of your existing uses of proc_role to has_role.

• has_role, a system function, checks whether an invoking user has been granted, and has activated, the specified role.

• has_role returns 0 if the user has:
  • Not been granted the specified role
  • Not been granted a role which contains the specified role
  • Been granted, but has not activated, the specified role

• has_role returns 1 if the invoking user has been granted, and has activated, the specified role.

• has_role returns 2 if the invoking user has a currently active role, which contains the specified role.
For general information about system functions, see “System functions” on page 77.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute `has_role`.

**See also**

**Commands**
- alter role
- create role
- drop role
- grant
- set
- revoke

**Functions**
- mut_excl_roles
- role_contain
- role_id
- role_name
- show_role
hash

Description
Produces a fixed-length hash value expression.

Syntax
hash(expression, [algorithm])

Parameters
- **expression** is the value to be hashed. This can be a column name, variable, constant expression, or any combination of these that evaluates to a single value. It cannot be image, text, unitext, or off-row Java datatypes. Expression is usually a column name. If expression is a character constant, it must be enclosed in quotes.
- **algorithm** is the algorithm used to produce the hash value. A character literal (not a variable or column name) that can take the values of either md5 or sha1, 2 (meaning md5 binary), or 3 (meaning sha1 binary). If omitted, md5 is used.

Examples
This example shows how a seal is implemented. The existence of a table called “atable” and with columns id, sensitive_field and tamper seal.

```
update atable set tamper_seal=hash(convert(varchar(30), id) + sensitive_field + @salt, 'sha1')
```

Usage
When specified as a character literal, **algorithm** is not case-sensitive—“md5”, “Md5” and “MD5” are equivalent. However, if **expression** is specified as a character datatype then the value is case sensitive. “Time,” “TIME,” and “time” will produce different hash values.

If **algorithm** is a character literal, the result is a varchar string. For “md5” this is a 32-byte string containing the hexadecimal representation of the 128-bit result of the hash calculation. For “sha1” this is a 40-byte string containing the hexadecimal representation of the 160-bit result of the hash calculation.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Results in</th>
</tr>
</thead>
</table>
| hash(expression, 'md5') | A varchar 32-byte string.
| md5 (Message Digest Algorithm 5) is the cryptographic hash function with a 128-bit hash value. |
| hash(expression) | A varchar 32-byte string |
| hash(expression, 'sha1') | A varchar 40-byte string
| sha1 (Secure Hash Algorithm) is the cryptographic hash function with a 160-bit hash value. |
| hash(expression, 2) | A varbinary 16-byte value (using the md5 algorithm) |
| hash(expression, 3) | A varbinary 20-byte value (using the sha1 algorithm) |
If *algorithm* is an integer literal, the result is a *varbinary* value. For 2, this is a 16-byte value containing the 128-bit result of the hash calculation. For 3, this is a 20-byte value containing the 160-bit result of the hash calculation.

**Note** Trailing null values are trimmed by Adaptive Server when inserted into *varbinary* columns.

Individual bytes that form *expression* are fed into the hash algorithm in the order they appear in memory. For many datatypes order is significant. For example, the binary representation of the 4-byte INT value 1 will be 0x00, 0x00, 0x00, 0x01 on MSB-first (big-endian) platforms and 0x01, 0x00, 0x00, 0x00 on LSB-first (little-endian) platforms. Because the stream of bytes is different between platforms, the hash value is different as well. Use `hashbytes` function to achieve platform independent hash value.

**Note** The hash algorithms MD5 and SHA1 are no longer considered entirely secure by the cryptographic community. As for any such algorithm, you should be aware of the risks of using MD5 or SHA1 in a security-critical context.

<table>
<thead>
<tr>
<th>Standards</th>
<th>SQL92- and SQL99- compliant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Permissions</td>
<td>Any user can execute hash.</td>
</tr>
<tr>
<td>See also</td>
<td>See also <code>hashbytes</code> for platform independent hash values.</td>
</tr>
</tbody>
</table>
hashbytes

Description

Produces a fixed-length, hash value expression.

Syntax

hashbytes(algorithm, expression[, expression...] [using options])

Parameters

expression[, expression...]

– is the value to be hashed. This value can be a column name, variable, constant expression, or a combination of these that produces a single value. It cannot be image, text, unitext, or off-row Java datatypes.

algorithm

is the algorithm used to produce the hash value. A character literal (not a variable or a column name) that can take the values “md5”, “sha”, “sha1”, or “ptn”.

Algorithm | Description
--- | ---
Md5 | Message Digest Algorithm 5 – is the cryptographic hash algorithm with a 128 bit hash value. hashbytes('md5', expression[...]) results in a varbinary 16-byte value.
Sha-1 | Secure Hash Algorithm – is the cryptographic hash algorithm with a 160-bit hash value. hashbytes('sha1', expression[...]) results in a varbinary 20-byte value.
Ptfn | The partition hash algorithm with 32-bit hash value. The using clause is ignored for the ‘ptn’ algorithm. hashbytes('ptn', expression[...]) results in an unsigned int 4-byte value.

using

Orders bytes for platform independence. The optional using clause can precede the following option strings:

• lsb – all byte-order dependent data is normalized to little-endian byte-order before being hashed.
• msb – all byte-order dependent data is normalized to big-endian byte-order before being hashed.
• unicode – character data is normalized to unicode (UTF–16) before being hashed.

Note A UTF–16 string is similar to an array of short integers. Because it is byte-order dependent, Sybase suggest for platform independence you use lsb or msb in conjunction with UNICODE.

• unicode_lsb – a combination of unicode and lsb.
• unicode_msb – a combination of unicode and msb.

Example 1 Seals each row of a table against tampering. This example assumes the existence of a user table called “xtable” and col1, col2, col3 and tamper_seal.
**hashbytes**

```sql
update xtable set tamper_seal=hashbytes('sha1', col1, col2, col4, @salt)
```
```
--
declare @nparts unsigned int
select @nparts= 5
select hashbytes('ptn', col1, col2, col3) % nparts from xtable
```

**Example 2** Shows how col1, col2, and col3 will be used to partition rows into five partitions.

```sql
alter table xtable partition by hash(col1, col2, col3) 5
```

**Usage**

The algorithm parameter is not case-sensitive; “md5,” “Md5” and “MD5” are all equivalent. However, if the expression is specified as a character datatype, the value is case sensitive. “Time,” “TIME,” and “time” will produce different hash values.

**Note** Trailing null values are trimmed by Adaptive Server when inserting into varbinary columns.

In the absence of a using clause, the bytes that form expression are fed into the hash algorithm in the order they appear in memory. For many datatypes, order is significant. For example, the binary representation of the 4-byte INT value 1 will be 0x00, 0x00, 0x00, 0x01, on MSB-first (big-endian) platforms and 0x01, 0x00, 0x00, 0x00 on LSB-first (little-endian) platforms. Because the stream of bytes is different for different platforms, the hash value is different as well.

With the using clause, the bytes that form expression can be fed into the hashing algorithm in a platform-independent manner. The using clause can also be used to transform character data into Unicode so that the hash value becomes independent of the server’s character configuration.

**Note** The hash algorithms MD5 and SHA1 are no longer considered entirely secure by the cryptographic community. Be aware of the risks of using MD5 or SHA1 in a security-critical context.

**Standards**

SQL92- and SQL99-compliant

**Permissions**

Any user can execute hashbyte.

**See also**

See also hash for platform dependent hash values.
**hextobigint**

**Description**
Returns the bigint value equivalent of a hexadecimal string

**Syntax**
hextobigint(hexadecimal_string)

**Parameters**
- **hexadecimal_string**
  is the hexadecimal value to be converted to a big integer; must be a character-type column, variable name, or a valid hexadecimal string, with or without a “0x” prefix, enclosed in quotes.

**Examples**
The following example converts the hexadecimal string 0x7fffffffffffffff to a big integer.

1> select hextobigint("0x7fffffffffffffff")
2> go
-------------------
9223372036854775807

**Usage**
- hextobigint, a datatype conversion function, returns the platform-independent integer equivalent of a hexadecimal string.
- Use the hextobigint function for platform-independent conversions of hexadecimal data to integers. hextobigint accepts a valid hexadecimal string, with or without a “0x” prefix, enclosed in quotes, or the name of a character-type column or variable.

hextobigint returns the bigint equivalent of the hexadecimal string. The function always returns the same bigint equivalent for a given hexadecimal string, regardless of the platform on which it is executed.

**See also**
- **Functions**
  biginttohex, convert, inttohex, hextoint
### hextoint

**Description**
Returns the platform-independent integer equivalent of a hexadecimal string.

**Syntax**
```
hextoint(hexadecimal_string)
```

**Parameters**
- `hexadecimal_string` is the hexadecimal value to be converted to an integer; must be a character-type column, variable name, or a valid hexadecimal string, with or without a “0x” prefix, enclosed in quotes.

**Examples**
Returns the integer equivalent of the hexadecimal string “0x00000100”. The result is always 256, regardless of the platform on which it is executed:
```
select hextoint ('0x00000100')
```

**Usage**
- `hextoint`, a datatype conversion function, returns the platform-independent integer equivalent of a hexadecimal string.

- Use the `hextoint` function for platform-independent conversions of hexadecimal data to integers. `hextoint` accepts a valid hexadecimal string, with or without a “0x” prefix, enclosed in quotes, or the name of a character-type column or variable.

  `hextoint` returns the integer equivalent of the hexadecimal string. The function always returns the same integer equivalent for a given hexadecimal string, regardless of the platform on which it is executed.

- For more information about datatype conversion, see “Datatype conversion functions” on page 63.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute `hextoint`.

**See also**
- **Functions** `bigintohex`, `convert`, `intohex`
**host_id**

**Description**

Returns the client computer’s operating system process ID for the current Adaptive Server client.

**Syntax**

host_id()

**Parameters**

None.

**Examples**

In this example, the name of the client computer is “ephemeris” and the process ID on the computer “ephemeris” for the Adaptive Server client process is 2309:

```
select host_name(), host_id()
```

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ephemeris</td>
<td>2309</td>
</tr>
</tbody>
</table>

The following is the process information, gathered using the UNIX `ps` command, from the computer “ephemeris” showing that the client in this example is “isql” and its process ID is 2309:

```
2309 pts/2 S 0:00 /work/as125/OCS-12_5/bin/isql
```

**Usage**

- `host_id`, a system function, returns the host process ID of the client process (not the server process).
- For general information about system functions, see “String functions” on page 76.

**Standards**

ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**

Any user can execute `host_id`.

**See also**

Function `host_name`
### host_name

**Description**
Returns the current host computer name of the client process.

**Syntax**
`host_name()`

**Parameters**
None.

**Examples**
```
select host_name()
```
```
-----------------------------
violet
```

**Usage**
- `host_name`, a system function, returns the current host computer name of the client process (not the server process).
- For general information about system functions, see “System functions” on page 77.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute `host_name`.

**See also**
- **Function** [host_id](#)
**instance_id**

**Description**  
Cluster environments only – Returns the id of the named instance, or the instance from which it is issued if you do not provide a value for *name*.

**Syntax**  
```
instance_id([name])
```

**Parameters**  
*name*  
name of the instance whose ID you are researching.

**Examples**  
Returns the ID of the local instance:

```
select instance_id()
```

Returns the ID of the instance named “myserver1”:

```
select instance_id(myserver1)
```

**Usage**  
Returns the ID of the instance named “myserver1”:

```
select instance_id(myserver1)
```

**Standards**  
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**  
Any user can execute `instance_id`.
### identity_burn_max

**Description**
Tracks the identity burn max value for a given table. This function returns only the value; does not perform an update.

**Syntax**
```sql
identity_burn_max(table_name)
```

**Parameters**
- `table_name` is the name of the table selected.

**Examples**
```sql
select identity_burn_max("t1")
```

```
t1
   --------
   51
```

**Usage**
`identity_burn_max` tracks the identity burn max value for a given table.

**Permissions**
Only the table owner, System Administrator, or database administrator can issue this command.
**index_col**

Description

Returns the name of the indexed column in the specified table or view, and can be up to 255 bytes in length.

Syntax

```
index_col(object_name, index_id, key_#, user_id)
```

Parameters

- `object_name` is the name of a table or view. The name can be fully qualified (that is, it can include the database and owner name). It must be enclosed in quotes.

- `index_id` is the number of `object_name`’s index. This number is the same as the value of `sysindexes.indid`.

- `key_#` is a key in the index. This value is between 1 and `sysindexes.keycnt` for a clustered index and between 1 and `sysindexes.keycnt+1` for a nonclustered index.

- `user_id` is the owner of `object_name`. If you do not specify `user_id`, it defaults to the caller’s user ID.

Examples

Finds the names of the keys in the clustered index on table `t4`:

```
declare @keycnt integer
select @keycnt = keycnt from sysindexes
    where id = object_id("t4")
    and indid = 1
while @keycnt > 0
begin
    select index_col("t4", 1, @keycnt)
    select @keycnt = @keycnt - 1
end
```

Usage

- `index_col`, a system function, returns the name of the indexed column.

- `index_col` returns NULL if `object_name` is not a table or view name.

- For general information about system functions, see “String functions” on page 76.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute `index_col`.

See also

- **Function** `object_id`
- **System procedure** `sp_helpindex`
index_colorder

**Description**
Returns the column order.

**Syntax**
```
index_colorder(object_name, index_id, key_, [user_id])
```

**Parameters**
- `object_name`
is the name of a table or view. The name can be fully qualified (that is, it can include the database and owner name). It must be enclosed in quotes.
- `index_id`
is the number of `object_name`'s index. This number is the same as the value of `sysindexes.indid`.
- `key_`
is a key in the index. Valid values are 1 and the number of keys in the index. The number of keys is stored in `sysindexes.keycnt`.
- `user_id`
is the owner of `object_name`. If you do not specify `user_id`, it defaults to the caller's user ID.

**Examples**
Returns “DESC” because the `salesind` index on the `sales` table is in descending order:

```
select name, index_colorder("sales", indid, 2)
from sysindexes
where id = object_id ("sales")
and indid > 0
```

**Usage**
- `index_colorder`, a system function, returns “ASC” for columns in ascending order or “DESC” for columns in descending order.
- `index_colorder` returns NULL if `object_name` is not a table name or if `key_` is not a valid key number.
- For general information about system functions, see “String functions” on page 76.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute `index_colorder`.
**index_name**

**Description**
Returns an index name, when you provide the index ID, the database ID, and the object on which the index is defined.

**Syntax**
```sql
index_name(dbid, objid, indid)
```

**Parameters**
- `dbid` is the ID of the database on which the index is defined.
- `objid` is the ID of the table (in the specified database) on which the index is defined.
- `indid` is the ID of the index for which you want a name.

**Examples**

**Example 1**
Illustrates the normal usage of this function.
```sql
select index_name(db_id("testdb"), object_id("testdb..tab_apl"),1)
```

**Example 2**
Illustrates the output if the database ID is NULL and you use the current database ID.
```sql
select index_name(NULL,object_id("testdb..tab_apl"),1)
```

**Example 3**
Displays the table name if the index ID is 0, and the database ID and object ID are valid.
```sql
select index_name(db_id("testdb"), object_id("testdb..tab_apl"),1)
```

**Usage**
- `index_name` uses the current database ID, if you pass a NULL value in the `dbid` parameter.
- `index_name` returns NULL if you pass a NULL value in the `dbid` parameter.
- `index_name` returns the object name, if the index ID is 0, and you pass valid inputs for the object ID and the database ID.

**Permissions**
Any user can execute this function.

**See also**
`db_id`, `object_id`
### inttohex

**Description**
Returns the platform-independent hexadecimal equivalent of the specified integer.

**Syntax**
\[
\text{inttohex}(\text{integer_expression})
\]

**Parameters**
- `integer_expression` is the integer value to be converted to a hexadecimal string.

**Examples**
```
select inttohex (10)
--------
0000000A
```

**Usage**
- `inttohex`, a datatype conversion function, returns the platform-independent hexadecimal equivalent of an integer, without a “0x” prefix.
- Use the `inttohex` function for platform-independent conversions of integers to hexadecimal strings. `inttohex` accepts any expression that evaluates to an integer. It always returns the same hexadecimal equivalent for a given expression, regardless of the platform on which it is executed.
- For more information about datatype conversion, see “Datatype conversion functions” on page 63.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute `inttohex`.

**See also**
- **Functions** convert, hextobigint, hextoint
<table>
<thead>
<tr>
<th><strong>Function</strong></th>
<th><strong>Description</strong></th>
<th><strong>Syntax</strong></th>
<th><strong>Parameters</strong></th>
<th><strong>Examples</strong></th>
<th><strong>Usage</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>isdate</td>
<td>Determines whether an input expression is a valid datetime value.</td>
<td>isdate(character_expression)</td>
<td>character_expression is a character-type variable, constant expression, or column name.</td>
<td>Example 1 Determines if the string 12/21/2005 is a valid datetime value: select isdate('12/21/2005')</td>
<td>Returns 1 if the expression is a valid datetime value; returns 0 if it is not. Returns 0 for NULL input.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Example 2 Determines if stor_id and date in the sales table are valid datetime values: select isdate(stor_id), isdate(date) from sales</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>store_id is not a valid datetime value, but date is.</td>
<td></td>
</tr>
</tbody>
</table>
**isnumeric**

**Description**
Determines if an expression is a valid numeric datatype.

**Syntax**

\[
\text{isnumeric (character_expression)}
\]

**Parameters**

- `character_expression` is a character-type variable, constant expression, or a column name.

**Examples**

- **Example 1** Determines if the values in the `postalcode` column of the `authors` table contains valid numeric datatypes:

  \[
  \text{select isnumeric(postalcode) from authors}
  \]

- **Example 2** Determines if the value $100.12345 is a valid numeric datatype:

  \[
  \text{select isnumeric("$100.12345")}
  \]

**Usage**

- Returns 1 if the input expression is a valid integer, floating point number, money or decimal type; returns 0 if it does not or if the input is a NULL value. A return value of 1 guarantees that you can convert the expression to one of these numeric types.
- You can include currency symbols as part of the input.
**is_quiesced**

**Description**
Indicates whether a database is in quiesce database mode. `is_quiesced` returns 1 if the database is quiesced and 0 if it is not.

**Syntax**
```
is_quiesced(dbid)
```

**Parameters**
- `dbid` is the database ID of the database.

**Examples**

**Example 1** Uses the test database, which has a database ID of 4, and which is not quiesced:
```
1> select is_quiesced(4)
2> go

--------
0

(1 row affected)
```

**Example 2** Uses the test database after running `quiesce database` to suspend activity:
```
1> quiesce database tst hold test
2> go
1> select is_quiesced(4)
2> go

--------
1

(1 row affected)
```

**Example 3** Uses the test database after resuming activity using `quiesce database`:
```
1> quiesce database tst release
2> go
1> select is_quiesced(4)
2> go

--------
0

(1 row affected)
```

**Example 4** Executes a `select` statement with `is_quiesced` using an invalid database ID:
```
1> select is_quiesced(-5)
```
is_quiesced

2> go

----------
    NULL

(1 row affected)

Usage

- is_quiesced has no default values. You see an error if you execute is_quiesced without specifying a database.
- is_quiesced returns NULL if you specify a database ID that does not exist.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute is_quiesced.

See also

Command quiesce database
**is_sec_service_on**

Description

Returns 1 if the security service is active and 0 if it is not.

Syntax

```sql
is_sec_service_on(security_service_nm)
```

Parameters

**security_service_nm**

is the name of the security service.

Examples

```sql
select is_sec_service_on("unifiedlogin")
```

Usage

- Use `is_sec_service_on` to determine whether a given security service is active during the session.
- To find valid names of security services, execute:

  ```sql
  select * from syssecmechs
  ```

  The result might look something like:

<table>
<thead>
<tr>
<th>sec_mech_name</th>
<th>available_service</th>
</tr>
</thead>
<tbody>
<tr>
<td>dce</td>
<td>unifiedlogin</td>
</tr>
<tr>
<td>dce</td>
<td>mutualauth</td>
</tr>
<tr>
<td>dce</td>
<td>delegation</td>
</tr>
<tr>
<td>dce</td>
<td>integrity</td>
</tr>
<tr>
<td>dce</td>
<td>confidentiality</td>
</tr>
<tr>
<td>dce</td>
<td>detectreplay</td>
</tr>
<tr>
<td>dce</td>
<td>detectseq</td>
</tr>
</tbody>
</table>

  The `available_service` column displays the security services that are supported by Adaptive Server.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute `is_sec_service_on`.

See also

*Function* show_sec_services
**isnull**

Description
Substitutes the value specified in expression2 when expression1 evaluates to NULL.

Syntax
isnull(expression1, expression2)

Parameters

expression

is a column name, variable, constant expression, or a combination of any of these that evaluates to a single value. It can be of any datatype, including unichar. expression is usually a column name. If expression is a character constant, it must be enclosed in quotes.

Examples
Returns all rows from the titles table, replacing null values in price with 0:
```
select isnull(price, 0)
from titles
```

Usage

- isnull, a system function, substitutes the value specified in expression2 when expression1 evaluates to NULL. For general information about system functions, see “String functions” on page 76.
- The datatypes of the expressions must convert implicitly, or you must use the convert function.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute isnull.

See also

Function convert
**isnumeric**

<table>
<thead>
<tr>
<th>Description</th>
<th>Determines if an expression is a valid numeric datatype.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td><code>isnumeric (character_expression)</code></td>
</tr>
<tr>
<td>Parameters</td>
<td><code>character_expression</code> is a character-type variable, constant expression, or a column name.</td>
</tr>
</tbody>
</table>
| Examples             | **Example 1** Determines if the values in the `postalcode` column of the `authors` table contains valid numeric datatypes:  
                        `select isnumeric(postalcode) from authors`  
                        **Example 2** Determines if the value $100.12345 is a valid numeric datatype:  
                        `select isnumeric("$100.12345")` |
| Usage                | • Returns 1 if the input expression is a valid integer, floating point number, money or decimal type; returns 0 if it does not or if the input is a NULL value. A return value of 1 guarantees that you can convert the expression to one of these numeric types.  
                        • You can include currency symbols as part of the input. |
### instance_name

**Description**  
Cluster environments only – Returns the name for the Adaptive Server whose id you provide, or the name of the Adaptive Server from which it is issued if you do not provide a value for id.

**Syntax**  
instance_name([id])

**Parameters**  
- **id**  
  is the ID of the Adaptive Server whose name you are researching.

**Examples**  
Returns the name of the instance with an ID of 12:

```sql  
select instance_name(12)  
```

**Standards**  
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**  
Any user can execute instance_name.
**lc_id**

**Description**
Cluster environments only – Returns the ID of the logical cluster whose name you provide, or the current logical cluster if you do not provide a name.

**Syntax**
lc_id(logical_cluster_name)

**Parameters**
- `logical_cluster_name` is the name of the logical cluster.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute `lc_id`
lc_name

Description
Cluster environments only – Returns the name of the logical cluster whose id you provide, or the current logical cluster if you do not provide an ID.

Syntax
lc_name([logical_cluster_ID])

Parameters
logical_cluster_ID
is the ID of the logical cluster.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute lc_name.
**Ict_admin**

**Description**
Manages the last-chance threshold, returns the current value of the last-chance threshold (LCT), and aborts transactions in a transaction log that has reached its LCT.

**Syntax**
```sql
Ict_admin({"lastchance" | "logfull" | "reserved_for_rollbacks"},
database_id
  |"reserve", {log_pages | 0 }
  | "abort", process-id [, database-id])
```

**Parameters**
- **lastchance**
  - creates a LCT in the specified database.
- **logfull**
  - returns 1 if the LCT has been crossed in the specified database and 0 if it has not.
- **reserved_for_rollbacks**
  - determines the number of pages a database currently reserved for rollbacks.
- **database_id**
  - specifies the database.
- **reserve**
  - obtains either the current value of the LCT or the number of log pages required for dumping a transaction log of a specified size.
- **log_pages**
  - is the number of pages for which to determine a LCT.
- **0**
  - returns the current value of the LCT. The size of the LCT in a database with separate log and data segments does not vary dynamically. It has a fixed value, based on the size of the transaction log. The LCT varies dynamically in a database with mixed log and data segments.
- **abort**
  - aborts transactions in a database where the transaction log has reached its last-chance threshold. Only transactions in log-suspend mode can be aborted.
- **logsegment_freepages**
  - describes the free space available for the log segment. This is the total value of free space, not per-disk.
process-id
is the ID (spid) of a process in log-suspend mode. A process is placed in
log-suspend mode when it has open transactions in a transaction log that has
reached its last-chance threshold (LCT).

database-id
is the ID of a database with a transaction log that has reached its LCT. If
process-id is 0, all open transactions in the specified database are
terminated.

Examples

Example 1 Creates the log segment last-chance threshold for the database with
dbid 1. It returns the number of pages at which the new threshold resides. If
there was a previous last-chance threshold, it is replaced:

```sql
select lct_admin("lastchance", 1)
```

Example 2 Returns 1 if the last-chance threshold for the database with dbid of
6 has been crossed, and 0 if it has not:

```sql
select lct_admin("logfull", 6)
```

Example 3 Calculates and returns the number of log pages that would be
required to successfully dump the transaction log in a log containing 64 pages:

```sql
select lct_admin("reserve", 64)
```

Example 4 Returns the current last-chance threshold of the transaction log in
the database from which the command was issued:

```sql
select lct_admin("reserve", 0)
```

Example 5 Aborts transactions belonging to process 83. The process must be
in log-suspend mode. Only transactions in a transaction log that has reached its
LCT are terminated:

```sql
select lct_admin("abort", 83)
```

Example 6 Aborts all open transactions in the database with dbid of 5. This
form awakens any processes that may be suspended at the log segment
last-chance threshold:

```sql
select lct_admin("abort", 0, 5)
```

Example 7 Determines the number of pages reserved for rollbacks in the
pubs2 database, which has a dbid of 5:

```sql
select lct_admin("reserved_for_rollbacks", 5, 0)
```

Example 8 Describes the free space available for a database with a dbid of 4:


select lct_admin("logsegment_freetables", 4)

**Usage**

- `lct_admin`, a system function, manages the log segment’s last-chance threshold. For general information about system functions, see “System functions” on page 77.

- If `lct_admin("lastchance", dbid)` returns zero, the log is not on a separate segment in this database, so no last-chance threshold exists.

- Whenever you create a database with a separate log segment, the server creates a default last chance threshold that defaults to calling `sp_thresholdaction`. This happens even if a procedure called `sp_thresholdaction` does not exist on the server at all.

  If your log crosses the last-chance threshold, Adaptive Server suspends activity, tries to call `sp_thresholdaction`, finds it does not exist, generates an error, then leaves processes suspended until the log can be truncated.

- To terminate the oldest open transaction in a transaction log that has reached its LCT, enter the ID of the process that initiated the transaction.

- To terminate all open transactions in a transaction log that has reached its LCT, enter 0 as the `process-id`, and specify a database ID in the `database-id` parameter.

**Standards**

ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**

Only a System Administrator can execute `lct_admin abort`. Any user can execute the other `lct_admin` options.

**See also**

- Document  *System Administration Guide.*
- Command  `dump transaction`
- Function  `curunreservedpgs`
- System procedure  `sp_thresholdaction`
left

Description

Returns a specified number of characters on the left end of a character string.

Syntax

`left(character_expression, integer_expression)`

Parameters

- `character_expression` is the character string from which the characters on the left are selected.
- `integer_expression` is the positive integer that specifies the number of characters returned. An error is returned if `integer_expression` is negative.

Examples

**Example 1** Returns the five leftmost characters of each book title.

```
use pubs
select left(title, 5) from titles
order by title_id
```

```
-----
The B
Cooki
You C
.....
Sushi
```

(18 row(s) affected)

**Example 2** Returns the two leftmost characters of the character string “abcdef”:

```
select left("abcdef", 2)
```

```
-------
ab
```

(1 row(s) affected)

Usage

- `character_expression` can be of any datatype (except text or image) that can be implicitly converted to varchar or nvarchar. `character_expression` can be a constant, variable, or a column name. You can explicitly convert `character_expression` using `convert`.

- `left` is equivalent to `substring(character_expression, 1, integer_expression)`. For more information on this function, see `substring` on page 297.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute `left`.

See also

Datatypes  varchar, nvarchar

Functions  len, str_replace, substring
### len

**Description**
Returns the number of characters, not the number of bytes, of a specified string expression, excluding trailing blanks.

**Syntax**
`len(string_expression)`

**Parameters**
- **string_expression** is the string expression to be evaluated.

**Examples**
Returns the characters
```
select len(notes) from titles
where title_id = "PC9999"
----------
39
```

**Usage**
This function is the equivalent of `char_length(string_expression)`.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute `len`.

**See also**
- **Datatypes** `char, nchar, varchar, nvarchar`
- **Functions** `char_length, left, str_replace`
license_enabled

**Description**
Returns 1 if a feature’s license is enabled, 0 if the license is not enabled, or NULL if you specify an invalid license name.

**Syntax**
```
license_enabled("ase_server" | "ase_ha" | "ase_dtm" | "ase_java" | "ase_asm")
```

**Parameters**
- **ase_server**
  specifies the license for Adaptive Server.
- **ase_ha**
  specifies the license for the Adaptive Server high availability feature.
- **ase_dtm**
  specifies the license for Adaptive Server distributed transaction management features.
- **ase_java**
  specifies the license for the Java in Adaptive Server feature.
- **ase_asm**
  specifies the license for Adaptive Server advanced security mechanism.

**Examples**
Indicates that the license for the Adaptive Server distributed transaction management feature is enabled:
```
select license_enabled("ase_dtm")
```

**Usage**
- For information about installing license keys for Adaptive Server features, see your installation guide.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute `license_enabled`.

**See also**
- **Documents**  Installation guide for your platform
- **System procedure**  `sp_configure`
list_appcontext

Description
Lists all the attributes of all the contexts in the current session. list_appcontext is a built-in function provided by the Application Context Facility (ACF).

Syntax
list_appcontext(["context_name"])

Parameters
context_name
is an optional argument that names all the application context attributes in the session.

Examples
Example 1 Shows the results when a user with appropriate permissions attempts to list the application contexts:

```sql
select list_appcontext ([context_name])
```

Output:
```
Context Name: (CONTEXT1)
  Attribute Name: (ATTR1) Value: (VALUE2)
Context Name: (CONTEXT2)
  Attribute Name: (ATTR1) Value: (VALUE1)
```

Example 2 Shows the results when a user without appropriate permissions attempts to list the application contexts:

```sql
select list_appcontext ()
```

Output:
```
Select permission denied on built-in list_appcontext,
database DBID --------
-1
```

Usage
- This function returns 0 for success.
- Since built-in functions do not return multiple result sets, the client application receives list_appcontext returns as messages.

Standards
ANSI SQL – Compliance level: Transact-SQL extension

Permissions
Permissions depend on the user profile and the application profile, and are stored by the ACF.

See also
For more information on the ACF, see “Row-level access control” in Chapter 11, “Managing User Permissions” of the System Administration Guide.

Functions
get_appcontext, list_appcontext, rm_appcontext, set_appcontext
**lockscheme**

**Description**
Returns the locking scheme of the specified object as a string.

**Syntax**

```
lockscheme(object_name)
lockscheme(object_id[, db_id])
```

**Parameters**
- `object_name` is the name of the object that the locking scheme returns. `object_name` can also be a fully qualified name.
- `db_id` is the ID of the database specified by `object_id`.
- `object_id` is the ID of the object that the locking scheme returns.

**Examples**

**Example 1** Selects the locking scheme for the `titles` table in the current database:
```
select lockscheme("titles")
```

**Example 2** Selects the locking scheme for `object_id` 224000798 (in this case, the `titles` table) from database ID 4 (the `pubs2` database):
```
select lockscheme(224000798, 4)
```

**Example 3** Returns the locking scheme for the `titles` table (`object_name` in this example is fully qualified):
```
select lockscheme(tempdb.ownerjoe.titles)
```

**Usage**
- `lockscheme` returns `varchar(11)` and allows NULLs.
- `lockscheme` defaults to the current database if you:
  - Do not provide a fully qualified `object_name`.
  - Do not provide a `db_id`.
  - Provide a null for `db_id`.
- If the specified object is not a table, `lockscheme` returns the string “not a table.”

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute `lockscheme`. 
log

Description
Returns the natural logarithm of the specified number.

Syntax
log(approx_numeric)

Parameters
approx_numeric
is any approximate numeric (float, real, or double precision) column name, variable, or constant expression.

Examples
select log(20)
--------------------
2.995732

Usage
• log, a mathematical function, returns the natural logarithm of the specified value.
• For general information about mathematical functions, see “Mathematical functions” on page 74.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute log.

See also
Functions log10, power
log10

Description
Returns the base 10 logarithm of the specified number.

Syntax
log10(approx_numeric)

Parameters
approx_numeric
is any approximate numeric (float, real, or double precision) column name, variable, or constant expression.

Examples
select log10(20)
------------------------
1.301030

Usage
• log10, a mathematical function, returns the base 10 logarithm of the specified value.
• For general information about mathematical functions, see “Mathematical functions” on page 74.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute log10.

See also
Functions log, power
lower

Description
Returns the lowercase equivalent of the specified expression.

Syntax
lower(char_expr | uchar_expr)

Parameters
char_expr
is a character-type column name, variable, or constant expression of char, varchar, nchar, or nvarchar type.

uchar_expr
is a character-type column name, variable, or constant expression of unichar or univarchar type.

Examples
select lower(city) from publishers

---------------------
boston
washington
berkeley

Usage
• lower, a string function, converts uppercase to lowercase, returning a character value.

• lower is the inverse of upper.

• If char_expr or uchar_expr is NULL, returns NULL.

• For general information about string functions, see “String functions” on page 76.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute lower.

See also
Function upper
**ltrim**

**Description**
Returns the specified expression, trimmed of leading blanks.

**Syntax**
ltrim(char_expr | uchar_expr)

**Parameters**
- **char_expr** is a character-type column name, variable, or constant expression of char, varchar, nchar, or nvarchar type.
- **uchar_expr** is a character-type column name, variable, or constant expression of unichar or univarchar type.

**Examples**
```
select ltrim(" 123")
--------
123
```

**Usage**
- ltrim, a string function, removes leading blanks from the character expression. Only values equivalent to the space character in the current character set are removed.
- If char_expr or uchar_expr is NULL, returns NULL.
- For Unicode expressions, returns the lowercase Unicode equivalent of the specified expression. Characters in the expression that have no lowercase equivalent are left unmodified.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute ltrim.

**See also**
For general information about string functions, see “String functions” on page 76.

**Function**
ltrim
**max**

**Description**
Returns the highest value in an expression.

**Syntax**
```sql
max(expression)
```

**Parameters**
*expression* is a column name, constant, function, any combination of column names, constants, and functions connected by arithmetic or bitwise operators, or a subquery.

**Examples**

**Example 1** Returns the maximum value in the *discount* column of the *salesdetail* table as a new column:

```sql
select max(discount) from salesdetail
```

```
--------------------
62.200000
```

**Example 2** Returns the maximum value in the *discount* column of the *salesdetail* table as a new row:

```sql
select discount from salesdetail
compute max(discount)
```

**Usage**

- *max*, an aggregate function, finds the maximum value in a column or expression. For general information about aggregate functions, see “Aggregate functions” on page 54.

- You can use *max* with exact and approximate numeric, character, and datetime columns; you cannot use it with bit columns. With character columns, *max* finds the highest value in the collating sequence. *max* ignores null values. *max* implicitly converts char datatypes to varchar, and unichar datatypes to univarchar, stripping all trailing blanks.

- *unichar* data is collated according to the default Unicode sort order.

- Adaptive Server goes directly to the end of the index to find the last row for *max* when there is an index on the aggregated column, unless:
  - The *expression* not a column.
  - The column is not the first column of an index.
  - There is another aggregate in the query.
  - There is a *group by* or *where* clause.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute *max*. 
See also

**Commands**  compute clause, group by and having clauses, select, where clause

**Functions**  avg, min
**min**

**Description**
Returns the lowest value in a column.

**Syntax**

```
min(expression)
```

**Parameters**

`expression` is a column name, constant, function, any combination of column names, constants, and functions connected by arithmetic or bitwise operators, or a subquery. With aggregates, an expression is usually a column name. For more information, see “Expressions” on page 359.

**Examples**

```
select min(price) from titles
  where type = "psychology"

------------------------
  7.00
```

**Usage**

- `min`, an aggregate function, finds the minimum value in a column.
- For general information about aggregate functions, see “Aggregate functions” on page 54.
- You can use `min` with numeric, character, time, and datetime columns; you cannot use it with bit columns. With character columns, `min` finds the lowest value in the sort sequence. `min` implicitly converts `char` datatypes to `varchar`, and `unichar` datatypes to `univarchar`, stripping all trailing blanks. `min` ignores null values. `distinct` is not available, since it is not meaningful with `min`.
- `unichar` data is collated according to the default Unicode sort order.
- Adaptive Server goes directly to the first qualifying row for `min` when there is an index on the aggregated column, unless:
  - The `expression` is not a column.
  - The column is not the first column of an index.
  - There is another aggregate in the query.
  - There is a group by clause.

**Standards**

ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**

Any user can execute `min`.

**See also**

- **Commands** compute clause, group by and having clauses, select, where clause
- **Functions** `avg`, `max`
### `month`

**Description**
Returns an integer that represents the month in the `datepart` of a specified date.

**Syntax**
```sql
month(date_expression)
```

**Parameters**
- `date_expression` is an expression of type `datetime`, `smalldatetime`, `date`, or a character string in a `datetime` format.

**Examples**
Returns the integer 11:
```

day("11/02/03")
----------
   11
```

**Usage**
`month(date_expression)` is equivalent to `datepart(mm, date_expression)`.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute `month`.

**See also**
- **Datatypes** `datetime`, `smalldatetime`, `date`
- **Functions** `datepart`, `day`, `year`
mut_excl_roles

Description
Returns information about the mutual exclusivity between two roles.

Syntax
mut_excl_roles (role1, role2 [membership | activation])

Parameters
role1
is one user-defined role in a mutually exclusive relationship.

role2
is the other user-defined role in a mutually exclusive relationship.

level
is the level (membership or activation) at which the specified roles are exclusive.

Examples
Shows that the admin and supervisor roles are mutually exclusive:
alter role admin add exclusive membership supervisor
select
mut_excl_roles("admin", "supervisor", "membership")
-----------
1

Usage
• mut_excl_roles, a system function, returns information about the mutual exclusivity between two roles. If the System Security Officer defines role1 as mutually exclusive with role2 or a role directly contained by role2, mut_excl_roles returns 1. If the roles are not mutually exclusive, mut_excl_roles returns 0.

Standards
ANSI SQL – Compliance level: Transact-SQL extension

Permissions
Any user can execute mut_excl_roles.

See also
For general information about system functions, see “System functions” on page 77.

Commands
alter role, create role, drop role, grant, set, revoke

Functions
proc_role, role_contain, role_id, role_name

System procedures
sp_activeroles, sp_displayroles, sp_role
**newid**

**Description**
Generates human-readable, globally unique IDs (GUIDs) in two different formats, based on arguments you provide. The length of the human-readable format of the GUID value is either 32 bytes (with no dashes) or 36 bytes (with dashes).

**Syntax**
newid([optionflag])

**Parameters**
- **option flag**
  - 0, or no value – the GUID generated is human-readable (varchar), but does not include dashes. This argument, which is the default, is useful for converting values into varbinary.
  - -1 – the GUID generated is human-readable (varchar) and includes dashes.
  - -0x0 – returns the GUID as a varbinary.
  - Any other value for newid returns NULL.

**Examples**

**Example 1** Creates a table with varchar columns 32 bytes long, then uses newid with no arguments with the insert statement:

```sql
create table t (UUID varchar(32))
go
insert into t values (newid())
insert into t values (newid())
go
select * from t
```

<table>
<thead>
<tr>
<th>UUID</th>
</tr>
</thead>
<tbody>
<tr>
<td>f81d4fae7dec11d0a76500a0c91e6bf6</td>
</tr>
<tr>
<td>7cd5b7769df75cefe040800208254639</td>
</tr>
</tbody>
</table>

**Example 2** Produces a GUID that includes dashes:

```sql
select newid(1)
```

<table>
<thead>
<tr>
<th>GUID</th>
</tr>
</thead>
<tbody>
<tr>
<td>b59462af-a55b-469d-a79f-1d6c3c1e19e3</td>
</tr>
</tbody>
</table>

**Example 3** Creates a default that converts the GUID format without dashes to a varbinary(16) column:

```sql
create table t (UUID_VC varchar(32), UUID varbinary(16))
go
create default default_guid as
```
Example 4  Returns a new GUID of type varbinary for every row that is returned from the query:

```sql
select newid(0x0) from sysobjects
```

Example 5  Uses newid with the varbinary datatype:

```sql
sp_addtype binguid, "varbinary(16)"
create default binguid_default as newid(0x0)
sp_bindefault "binguid_default","binguid"
create table T1 (empname char(60), empid int, emp_guid binguid)
insert T1 (empname, empid) values ("John Doe", 1)
insert T1 (empname, empid) values ("Jane Doe", 2)
```

Usage

- newid generates two values for the globally unique ID (GUID) based on arguments you pass to newid. The default argument generates GUIDs without dashes. By default newid returns new values for every filtered row.
- You can use newid in defaults, rules, and triggers, similar to other functions.
- Make sure the length of the varchar column is at least 32 bytes for the GUID format without dashes, and at least 36 bytes for the GUID format with dashes. The column length is truncated if it is not declared with these minimum required lengths. Truncation increases the probability of duplicate values.
- An argument of zero is equivalent to the default.
- You can use the GUID format without dashes with the strtoyin function to convert the GUID value to 16-byte binary data. However, using strtoyin with the GUID format with dashes results in NULL values.
- Because GUIDs are globally unique, they can be transported across domains without generating duplicates.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute newid.
next_identity

Description
Retrieves the next identity value that is available for the next insert.

Syntax
next_identity(table_name)

Parameters
- table_name identifies the table being used.

Examples
Updates the value of c2 to 10. The next available value is 11.

```sql
select next_identity ("t1")
t1
--------
11
```

Usage
- `next_identity` returns the next value to be inserted by this task. In some cases, if multiple users are inserting values into the same table, the actual value reported as the next value to be inserted is different from the actual value inserted if another user performs an intermediate insert.

- `next_identity` returns a `varchar` character to support any precision of the identity column. If the table is a proxy table, a non-user table, or the table does not have identity property, NULL is returned.

Permissions
Only the table owner, System Administrator, or database administrator can issue this command.
nullif

Description
Supports conditional SQL expressions; can be used anywhere a value expression can be used; alternative for a case expression.

Syntax
nullif(expression, expression)

Parameters
nullif
compares the values of the two expressions. If the first expression equals the second expression, nullif returns NULL. If the first expression does not equal the second expression, nullif returns the first expression.

expression
is a column name, a constant, a function, a subquery, or any combination of column names, constants, and functions connected by arithmetic or bitwise operators. For more information about expressions, see “Expressions” on page 359.

Examples
Example 1 Selects the titles and type from the titles table. If the book type is UNDECIDED, nullif returns a NULL value:

```
select title,
nullif(type, "UNDECIDED")
from titles
```

Example 2 This is an alternative way of writing Example 1:

```
select title,
case
   when type = "UNDECIDED" then NULL
   else type
end
from titles
```

Usage
• nullif expression alternate for a case expression.
• nullif expression simplifies standard SQL expressions by allowing you to express a search condition as a simple comparison instead of using a when...then construct.
• You can use nullif expressions anywhere an expression can be used in SQL.
• At least one result of the case expression must return a non-null value. For example the following results in an error message:

```
select price, coalesce (NULL, NULL, NULL)
from titles
```
All result expressions in a CASE expression must not be NULL.
nullif

- If your query produces a variety of datatypes, the datatype of a case expression result is determined by datatype hierarchy, as described in “Datatypes of mixed-mode expressions” on page 7. If you specify two datatypes that Adaptive Server cannot implicitly convert (for example, char and int), the query fails.

Standards  ANSI SQL – Compliance level: Transact-SQL extension.
Permissions Anyone can execute nullif.
See also  Commands case, coalesce, select, if...else, where clause
object_attr

Description
Reports the table’s current logging mode, depending on the session, table and database-wide settings.

Syntax
object_attr(table_name, string)

Parameters

- **table_name**
  name of a table.

- **string**
  is the name of the property of the table which is been queried for. The supported string values are:

  - **dml_logging** – returns the DML logging level for the requested object in effect based on the explicitly set table or database’s DML logging level.

  - **dml_logging for session** – returns the DML logging level for the current session, taking into account the user running object_attr, the table’s schema, and rules regarding multistatement transactions, and so on. The return value from this argument can be different for different users, and different for statements or transactions for the same user.

  - **help** – prints a list of supported string arguments.

Examples

**Example 1** To determine which properties they can query, the user runs:

```sql
select object_attr('sysobjects', 'help')
```

Usage: object_attr('tabname', 'attribute')

List of options in attributes table:

<table>
<thead>
<tr>
<th></th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>help</td>
</tr>
<tr>
<td>1</td>
<td>dml_logging</td>
</tr>
<tr>
<td>2</td>
<td>dml_logging for session</td>
</tr>
</tbody>
</table>

**Example 2** The default logging mode of a table with durability set to full:

```sql
select object_attr("pubs2..authors", "dml_logging")
```

Returns: FULL

**Example 3** If the session has logging disabled for all tables, the logging mode returned for tables owned by this user is minimal.

```sql
select object_attr("pubs2..authors",
```

Reference Manual: Building Blocks 223
object_attr

"dml_logging")
Returns: FULL

SET DML_LOGGING MINIMAL
go

select object_attr("pubs2..authors",
"dml_logging for session")
Returns: MINIMAL

**Example 4** If a table has been altered to explicitly select minimal logging, object_attr returns a value of minimal, even if the session and database-wide logging is FULL.

create database testdb WITH DML_LOGGING = FULL
go

create table non_logged_table (...) WITH DML_LOGGING = MINIMAL
go

select object_attr("non_logged_table",
"dml_logging")
Returns: MINIMAL

**Example 5** Changes a table’s logging from full to minimal. If you explicitly create a table with full logging, you can reset the logging to minimal during a session if you are the table owner or a user with the sa_role:

1 Create the testdb database with minimal logging:
   
   create database testdb
   with dml_logging = minimal

2 Create a table with dml logging set to full:
   
   create table logged_table(...) with dml_logging = full

3 Reset the logging for the session to minimal:
   
   set dml_logging minimal

4 The logging for the table is minimal:
   
   select object_attr("logged_table",
   "dml_logging for session")
--------------------
Example 6  If you create a table without specifying the logging mode, changing the session’s logging mode also changes the table’s logging mode:

- Create the table normal_table:
  ```sql
  create table normal_table
  ```

- Check the session’s logging:
  ```sql
  select object_attr("normal_table", "dml_logging")
  -------------------------
  FULL
  ```

- Set the session logging to minimal:
  ```sql
  set dml_logging minimal
  ```

- The table’s logging is set to minimal:
  ```sql
  select object_attr("normal_table", "dml_logging for session")
  -------------------------
  minimal
  ```

Example 7  The logging mode returned by `object_attr` depends on the table you run it against. In this example, user joe runs a script, but the logging mode Adaptive Server returns changes. The tables `joe.own_table` and `mary.other_table` use a full logging mode:

```sql
select object_attr("own_table", "dml_logging")
-------------------------
FULL
```

When joe runs `object_attr` against `mary.other_table`, this table is also set to full:

```sql
select object_attr("mary.other_table", "dml_logging")
-------------------------
FULL
```

If joe changes the `dml_logging` to minimal, only the logging mode of the tables he owns are affected:

```sql
set dml_logging minimal
select object_attr("own_table", "dml_logging for session")
-------------------------
MINIMAL
```

Tables owned by other users will continue to operate in their default logging mode:
Example 8  Identify the run-time choices of logging a new show_exec_info, and use it in the SQL batch:

1  Enable set showplan:
   
   set showplan on

2  Enable the set command:
   
   set show_exec_info on

3  Set dml_logging to minimal and check the logging with object_attr:
   
   set dml_logging minimal
   select object_attr("logged_table", "dml_logging for session")

4  Delete rows from the table:
   
   delete logged_table

Adaptive Server reports the table’s logging mode at run-time with show_exec_info parameter.

Usage

• The return type is a varchar, which appropriately returns the value of the property (for example, on or off) depending on the property queried for.

• The logging mode as reported by extensions to showplan output might be affected at run-time, if there are set statements in the same batch, preceding the execution of the DML, which changes the logging mode of the table

• The return value is the value NULL (not the string “NULL”) for an unknown property.

• A special-type of string parameter, help prints to the session’s output all the currently supported properties for object_attr. This allows you to quickly identify which properties are supported by object_attr.

Permissions

See also
**object_id**

**Description**
Returns the object ID of the specified object.

**Syntax**
```sql
object_id(object_name)
```

**Parameters**
- **object_name**
  is the name of a database object, such as a table, view, procedure, trigger, default, or rule. The name can be fully qualified (that is, it can include the database and owner name). Enclose the `object_name` in quotes.

**Examples**

**Example 1**
```sql
select object_id("titles")
```
```
-------------------
208003772
```  

**Example 2**
```sql
select object_id("master..sysobjects")
```
```
-------------
1
```  

**Usage**
`object_id`, a system function, returns the object’s ID. Object IDs are stored in the `id` column of `sysobjects`.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute `object_id`.

**See also**
For general information about system functions, see “System functions” on page 77.

**Functions**
- `col_name`, `db_id`, `object_name`

**System procedure**
- `sp_help`
**object_name**

Returns the name of the object with the object ID you specify; can be up to 255 bytes in length.

**Syntax**

```
object_name(object_id[, database_id])
```

**Parameters**

- `object_id` is the object ID of a database object, such as a table, view, procedure, trigger, default, or rule. Object IDs are stored in the `id` column of `sysobjects`.

- `database_id` is the ID for a database if the object is not in the current database. Database IDs are stored in the `db_id` column of `sysdatabases`.

**Examples**

**Example 1**

```
select object_name(208003772)
```

```
----------------------
titles
```

**Example 2**

```
select object_name(1, 1)
```

```
----------------------
sysobjects
```

**Usage**

`object_name`, a system function, returns the object’s name.

**Standards**

ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**

Any user can execute `object_name`.

**See also**

For general information about system functions, see “System functions” on page 77.

**Functions**

`col_name`, `db_id`, `object_id`

**System procedure**

`sp_help`
object_owner_id

Description
Returns an object’s owner ID.

Syntax
object_owner_id(object_id[, database_id])

Parameters
object_id
is the ID of the object you are investigating.

database_id
is the ID of the database in which the object resides.

Examples
Selects the owner’s ID for an object with an ID of 1, in the database with the ID of 1 (the master database):

          select object_owner_id(1,1)

Permissions
Any user can execute object_owner_id.
pagesize

**Description**
Returns the page size, in bytes, for the specified object.

**Syntax**
```sql
pagesize(object_name[,])
pagesize(object_id[,db_id[, index_id]])
```

**Parameters**
- `object_name` is the object name of the page size of this function returns.
- `index_name` indicates the index name of the page size you want returned.
- `object_id` is the object ID of the page size this function returns.
- `db_id` is the database ID of the object.
- `index_id` is the index ID of the object you want returned.

**Examples**

**Example 1** Selects the page size for the `title_id` index in the current database.
```
select pagesize("title", "title_id")
```

**Example 2** Returns the page size of the data layer for the object with `object_id` 1234 and the database with a `db_id` of 2 (the previous example defaults to the current database):
```
select pagesize(1234, null, 2)
select pagesize(1234, 2)
select pagesize(1234)
```

**Example 3** All default to the current database:
```
select pagesize(1234, null, 2)
select pagesize(1234)
```

**Example 4** Selects the page size for the `titles` table (`object_id` 224000798) from the `pubs2` database (`db_id` 4):
```
select pagesize(224000798, 4)
```

**Example 5** Returns the page size for the nonclustered index’s pages table `mytable`, residing in the current database:
```
pagesize(object_id('mytable'), NULL, 2)
```

**Example 6** Returns the page size for object `titles_clustindex` from the current database:
select pagesize("titles", "titles_clustindex")

Usage

- `pagesize` defaults to the data layer if you do not provide an index name or `index_id` (for example, `select pagesize("t1")`) if you use the word “null” as a parameter (for example, `select pagesize("t1", null)`).

- If the specified object is not an object requiring physical data storage for pages (for example, if you provide the name of a view), `pagesize` returns 0.

- If the specified object does not exist, `pagesize` returns NULL.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute `pagesize`.
partition_id

Description
Returns the partition ID of the specified data or index partition name.

Syntax
partition_id(table_name, partition_name[, index_name])

Parameters
- table_name
  is the name for a table.
- partition_name
  is the partition name for a table partition or an index partition.
- index_name
  is the name of the index of interest.

Examples
Example 1 Returns the partition ID corresponding to the partition name testtable_ptn1 and index id 0 (the base table). The testtable must exist in the current database:

```sql
select partition_id("testtable", "testtable_ptn1")
```

Example 2 Returns the partition ID corresponding to the partition name testtable_clust_ptn1 for the index name clust_index1. The testtable must exist in the current database:

```sql
select partition_id("testtable", "testtable_clust_ptn1", "clust_index1")
```

Example 3 This is the same as the previous example, except that the user need not be in the same database as where the target table is located:

```sql
select partition_id("mydb.dbo.testtable", "testtable_clust_ptn1", "clust_index1")
```

Usage
You must enclose table_name, partition_name and index_name in quotes.

See also
Functions data_pages, object_id, partition_name, reserved_pages, row_count, used_pages
**partition_name**

**Description**
The explicit name of a new partition, `partition_name` returns the partition name of the specified data or index partition id.

**Syntax**
```
partition_name(indid, ptnid[, dbid])
```

**Parameters**
- `indid` is the index ID for the target partition.
- `ptnid` is the ID of the target partition.
- `dbid` is the database ID for the target partition. If you do not specify this parameter, the target partition is assumed to be in the current database.

**Examples**

**Example 1** Returns the partition name for the given partition ID belonging to the base table (with an index ID of 0). The lookup is done in the current database because it does not specify a database ID:

```
select partition_name(0, 1111111111)
```

**Example 2** Returns the partition name for the given partition ID belonging to the clustered index (index ID of 1 is specified) in the `testdb` database.

```
select partition_name(1, 1212121212, db_id("testdb"))
```

**Usage**
- If the search does not find the target partition, the return is NULL.

**See also**
- Functions `data_pages`, `object_id`, `partition_id`, `reserved_pages`, `row_count`
**partition_object_id**

Description  Displays the object ID for a specified partition ID and database ID.

Syntax  `partition_object_id(partition_id [ , database_id ] )`

Parameters

- `partition_id` is the ID of the partition whose object ID is to be retrieved.
- `database_id` is the database ID of the partition.

Examples

**Example 1** Displays the object ID for the partition whose partition ID is 2:

```sql
select partition_object_id(2)
```

**Example 2** Displays the object ID for the partition whose partition ID is 14 and whose database ID is 7:

```sql
select partition_object_id(14,7)
```

**Example 3** Returns a NULL value for the database ID because a NULL value is passed to the function:

```sql
select partition_object_id( 1424005073, NULL)
```

Usage

- `partition_object_id` uses the current database ID if you do not include a database ID.
- `partition_object_id` returns NULL if you use a NULL value for the `partition_id`.
- `partition_object_id` returns a NULL value if you include a NULL value for database ID.
- `partition_object_id` returns NULL if you provide an invalid or non-existent `partition_id` or `database_id`. 
**patindex**

**Description**

Returns the starting position of the first occurrence of a specified pattern.

**Syntax**

```sql
patindex("%pattern%", char_expr|uchar_expr[, using {bytes | characters | chars}])
```

**Parameters**

- `pattern` is a character expression of the `char` or `varchar` datatype that may include any of the pattern-match wildcard characters supported by Adaptive Server. The `%` wildcard character must precede and follow `pattern` (except when searching for first or last characters). For a description of the wildcard characters, see “Pattern matching with wildcard characters” on page 377.

- `char_expr` is a character-type column name, variable, or constant expression of `char`, `varchar`, `nchar`, or `nvarchar` type.

- `uchar_expr` is a character-type column name, variable, or constant expression of `unichar`, or `univarchar` type.

- `using` specifies a format for the starting position.
  - `bytes` returns the offset in bytes.
  - `chars` or `characters` returns the offset in characters (the default).

**Examples**

**Example 1** Selects the author ID and the starting character position of the word “circus” in the `copy` column:

```sql
select au_id, patindex("%circus%", copy)
from blurbs
```

<table>
<thead>
<tr>
<th>au_id</th>
<th>Patindex Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>486-29-1786</td>
<td>0</td>
</tr>
<tr>
<td>648-92-1872</td>
<td>0</td>
</tr>
<tr>
<td>998-72-3567</td>
<td>38</td>
</tr>
<tr>
<td>899-46-2035</td>
<td>31</td>
</tr>
<tr>
<td>672-71-3249</td>
<td>0</td>
</tr>
<tr>
<td>409-56-7008</td>
<td>0</td>
</tr>
</tbody>
</table>

**Example 2**

```sql
select au_id, patindex("%circus%", copy,
```
Example 3  Finds all the rows in sysobjects that start with “sys” with a fourth character that is “a”, “b”, “c”, or “d”:

```sql
select name
from sysobjects
where patindex("sys[a-d] %", name) > 0
```

<table>
<thead>
<tr>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>sysalternates</td>
</tr>
<tr>
<td>sysattributes</td>
</tr>
<tr>
<td>syscharsets</td>
</tr>
<tr>
<td>syscolumns</td>
</tr>
<tr>
<td>syscomments</td>
</tr>
<tr>
<td>sysconfigures</td>
</tr>
<tr>
<td>syscurconfigs</td>
</tr>
<tr>
<td>sysdatabases</td>
</tr>
<tr>
<td>sysdepends</td>
</tr>
<tr>
<td>sysdevices</td>
</tr>
</tbody>
</table>

Usage

- `patindex`, a string function, returns an integer representing the starting position of the first occurrence of `pattern` in the specified character expression, or a 0 if `pattern` is not found.
- You can use `patindex` on all character data, including text and image data.
- For `unichar`, `univarchar`, and `unitext`, `patindex` returns the offset in Unicode characters. The pattern string is implicitly converted to UTF-16 before comparison, and the comparison is based on the default unicode sort order configuration. For example, this is what is returned if a `unitext` column contains row value U+0041U+0042U+d800U+dc00U+0043:

```sql
select patindex("%C%", ut) from unitable
```

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

- By default, `patindex` returns the offset in characters; to return the offset in bytes (multibyte character strings), specify using `bytes`.
- Include percent signs before and after `pattern`. To look for `pattern` as the first characters in a column, omit the preceding `. To look for `pattern` as the last characters in a column, omit the trailing `. 
- If `char_expr` or `uchar_expr` is NULL, `patindex` returns 0.
• If you give a varchar expression as one parameter and a unichar expression as the other, the varchar expression is implicitly converted to unichar (with possible truncation).

• For general information about string functions, see “String functions” on page 76.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute patindex.

See also Functions charindex, substring
**pi**

Description
Returns the constant value 3.1415926535897936.

Syntax
pi()

Parameters
None

Examples
```
select pi()
```

Usage
pi, a mathematical function, returns the constant value of 3.1415926535897931.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute pi.

See also
For general information about mathematical functions, see “Mathematical functions” on page 74.

Functions  
degrees, radians
power

Description
Returns the value that results from raising the specified number to a given power.

Syntax
`power(value, power)`

Parameters
`value`
- is a numeric value.

`power`
- is an exact numeric, approximate numeric, or money value.

Examples
```
select power(2, 3)

-----------

8
```

Usage
- `power`, a mathematical function, returns the value of `value` raised to the power `power`. Results are of the same type as `value`.

  In expressions of type `numeric` or `decimal`, this function returns precision:38, scale 18.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute `power`.

See also
For general information about mathematical functions, see “Mathematical functions” on page 74.

**Functions**` exp, log, log10`
**proc_role**

**Description**

Returns information about whether the user has been granted the specified role.

**Note** Sybase supports—and recommends—that you use `has_role` instead of `proc_role`. You need not, however, convert your existing uses of `proc_role` to `has_role`.

**Syntax**

```
proc_role("role_name")
```

**Parameters**

`role_name` is the name of a system or user-defined role.

**Examples**

**Example 1** Creates a procedure to check if the user is a System Administrator:

```sql
create procedure sa_check as
if (proc_role("sa_role") > 0)
begin
    print "You are a System Administrator."
    return(1)
end
```

**Example 2** Checks that the user has been granted the System Security Officer role:

```
select proc_role("sso_role")
```

**Example 3** Checks that the user has been granted the Operator role:

```
select proc_role("oper_role")
```

**Usage**

- Using `proc_role` with a procedure that starts with “sp_” returns an error.
- `proc_role`, a system function, checks whether an invoking user has been granted, and has activated, the specified role.
- `proc_role` returns 0 if the user has:
  - Not been granted the specified role
  - Not been granted a role which contains the specified role
  - Been granted, but has not activated, the specified role
- `proc_role` returns 1 if the invoking user has been granted, and has activated, the specified role.
- `proc_role` returns 2 if the invoking user has a currently active role, which contains the specified role.

**Standards**

ANSI SQL – Compliance level: Transact-SQL extension.
Permissions

Any user can execute proc_role.

See also

For general information about system functions, see “System functions” on page 77.

- **Commands**
  - alter role, create role, drop role, grant, set, revoke

- **Functions**
  - mut_excl_roles, role_contain, role_id, role_name, show_role
pssinfo

Description
Returns information from the Adaptive Server process status structure (pss).

Syntax
pssinfo(spid | 0, 'pss_field')

Parameters
spid
is the process ID. When you enter 0, the current process is used.

pss_field
is the process status structure field. Valid values are:
- dn – distinguished name when using LDAP authentication.
- extusername – when using external authentication like (PAM, LDAP), extusername returns the external PAM or LDAP user name used.
- ipaddr – client IP address.
- ipport – client IP port number used for the client connection associated with the user task being queried.
- isolation_level – isolation level for the current session.
- tempdb_pages – number of tempdb pages used.

Examples
Displays the port number for spid number 14
select pssinfo(14,'ipport')
----------------------
52039

Usage
- The pssinfo function also includes the option to display the external user name and the distinguish name.
- ipport output, combined with ipaddr output, allows you to uniquely identify network traffic between Adaptive Server and the client.
**radians**

**Description**
Returns the size, in radians, of an angle with the specified number of degrees.

**Syntax**
radians(numeric)

**Parameters**
numeric

is any exact numeric (numeric, dec, decimal, tinyint, smallint, or int), approximate numeric (float, real, or double precision), or money column, variable, constant expression, or a combination of these.

**Examples**

```sql
select radians(2578)
```

```
-----------
  44
```

**Usage**

- radians, a mathematical function, converts degrees to radians. Results are of the same type as numeric.

To express numeric or decimal datatypes, this function returns precision: 38, scale 18.

When money datatypes are used, internal conversion to float may cause loss of precision.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute radians.

**See also**
For general information about mathematical functions, see “Mathematical functions” on page 74.

**Function** degrees
rand

Description
Returns a random value between 0 and 1, which is generated using the specified seed value.

Syntax
rand(integer)

Parameters
integer is any integer (tinyint, smallint, or int) column name, variable, constant expression, or a combination of these.

Examples
Example 1
select rand()
--------------------
0.395740

Example 2
declare @seed int
select @seed=100
select rand(@seed)
--------------------
0.000783

Usage
• rand, a mathematical function, returns a random float value between 0 and 1, using the optional integer as a seed value.

• The rand function uses the output of a 32-bit pseudorandom integer generator. The integer is divided by the maximum 32-bit integer to give a double value between 0.0 and 1.0. The rand function is seeded randomly at server start-up, so getting the same sequence of random numbers is unlikely, unless the user first initializes this function with a constant seed value. The rand function is a global resource. Multiple users calling the rand function progress along a single stream of pseudorandom values. If a repeatable series of random numbers is needed, the user must assure that the function is seeded with the same value initially and that no other user calls rand while the repeatable sequence is desired.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute rand.

See also
For general information about mathematical functions, see “Mathematical functions” on page 74.

Datatypes Approximate numeric datatypes

Functions rand2
**rand2**

**Description**  
Returns a random value between 0 and 1, which is generated using the specified seed value, and computed for each returned row when used in the select list.

**Syntax**  
`rand2([integer])`

**Parameters**  
`integer`  
is any integer (tinyint, smallint, or int) column name, variable, constant expression, or a combination of these.

**Examples**  
If there are n rows is table t, the following select statement returns n different random values, not just one.

```sql
select rand2() from t
```

**Usage**  
- `rand2`, a mathematical function, returns a random float value between 0 and 1, using the optional integer as a seed value. Unlike `rand`, it is computed for each returned row when it is used in the select list.
- The behavior of `rand2` in places other than the select list is currently undefined.
- For more information about the 32-bit pseudorandom integer generator, see the Usage section of `rand`.

**Standards**  
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**  
Any user can execute `rand`.

**See also**  
For general information about mathematical functions, see “Mathematical functions” on page 74.

**Datatypes**  
Approximate numeric datatypes

**Functions**  
`rand`
**replicate**

**Description**

Returns a string consisting of the specified expression repeated a given number of times.

**Syntax**

```
replicate(char_expr | uchar_expr, integer_expr)
```

**Parameters**

- `char_expr` is a character-type column name, variable, or constant expression of `char`, `varchar`, `nchar`, or `nvarchar` type.
- `uchar_expr` is a character-type column name, variable, or constant expression of `unichar` or `univarchar` type.
- `integer_expr` is any integer (`tinyint`, `smallint`, or `int`) column name, variable, or constant expression.

**Examples**

```
select replicate("abcd", 3)
----------------
abcdabcdabcd
```

**Usage**

- `replicate`, a string function, returns a string with the same datatype as `char_expr` or `uchar_expr` containing the same expression repeated the specified number of times or as many times as fits into 16K, whichever is less.
- If `char_expr` or `uchar_expr` is NULL, returns a single NULL.

**Standards**

ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**

Any user can execute `replicate`.

**See also**

For general information about string functions, see “String functions” on page 76.

**Function**

`stuff`
reserve_identity

Description

reserve_identity allows a process to reserve a block of identity values for use by that process.

After a process calls reserve_identity to reserve the block of values, subsequent identity values needed by this process are drawn from this reserved pool. When these reserved numbers are exhausted, or if you insert data into a different table, the existing identity options apply. reserve_identity can retain more than one block of identity values, so if inserts to different tables are interleaved by a single process, the next value in a table’s reserved block is used.

Reserves a specified size block of identity values for the specified table, which are used exclusively by the calling process. Returns the reserved starting number, and subsequent inserts into the specified table by this process use these values. When the process terminates, any unused values are eliminated.

Syntax

`reserve_identity (table_name, number_of_values)`

Parameters

table_name

is the name of the table for which the reservation are made. The name can be fully qualified; that is, it can include the database_name, owner_name, and object_name (in quotes).

number_of_values

is the number of sequential identity values reserved for this process. This must be a positive value that will not cause any of the reserved values to exceed the maximum values for the datatype of the identity column.

Examples

Describes a typical usage scenario for reserve_identity, and assumes that table1 includes col1 (with a datatype of int) and a col2 (an identity column with a datatype of int). This process is for spid 3:

```sql
select reserve_identity( table1, 5 )
-----------------
10
```

Insert values for spids 3 and 4:

- Insert table1 values(56) -> spid 3
- Insert table1 values(48) -> spid 3
- Insert table1 values(96) -> spid 3
- Insert table1 values(02) -> spid 4
- Insert table1 values(84) -> spid 3

Select from table table1:

```sql
select * from table1
```
The result set shows that spid 3 reserved identity values 1–5, spid 4 receives the next unreserved value, and then spid 3 reserves the subsequent identity values.

Usage

- The return value, *start_value*, is the starting value for the block of reserved identity values. The calling process uses this value for the next insert into the specified table.
- `reserve_identity` allows a process to:
  - Reserve identity values without issuing an `insert` statement.
  - Know the values reserved prior issuing the `insert` statement.
  - “Grab” different size blocks of identity values, according to need.
  - Better control “over gaps” by reserving only what is needed (that is, they are not restricted by preset server grab size).
- Values are automatically used with no change to the `insert` syntax.
- NULL values are returned if:
  - A negative value or zero is specified as the block size.
  - The table does not exist.
  - The table does not contain an identity column.
- If you issue `reserve_identity` on a table in which this process has already reserved these identity values, the function succeeds and the most recent group of values is used.
- You cannot use `reserve_identity` to reserve identity values on a proxy table. Local servers can use `reserve_identity` on a remote table if the local server calls a remote procedure that calls `reserve_identity`. Because these reserved values are stored on the remote server but in the session belonging to the local server, subsequent inserts to the remote table use the reserved values.
If the identity_gap is less than the reserved block size, the reservation succeeds by reserving the specified block size (not an identity_gap size) of values. If these values are not used by the process, this results in potential gaps of up to the specified block size regardless of the identity_gap setting.

Permissions

You must have insert permission to reserve identity values.
reserved_pages

Description Reports the number of pages reserved for a database, object, or index. The result includes pages used for internal structures.

This function replaces the old reserved_pgs function used in Adaptive Server versions earlier than 15.0.

Syntax reserved_pages(dbid, object_id[, indid[, ptnid]])

Parameters dbid is the database ID of the database where the target object resides.

object_id is an object ID for a table.

indid is the index ID of target index.

ptnid is the partition ID of target partition.

Examples Example 1 Returns the number of pages reserved by the object with a object ID of 31000114 in the specified database (including any indexes):

select reserved_pages(5, 31000114)

Example 2 Returns the number of pages reserved by the object in the data layer, regardless of whether or not a clustered index exists:

select reserved_pages(5, 31000114, 0)

Example 3 Returns the number of pages reserved by the object in the index layer for a clustered index. This does not include the pages used by the data layer:

select reserved_pages(5, 31000114, 1)

Example 4 Returns the number of pages reserved by the object in the data layer of the specific partition, which in this case is 2323242432:

select reserved_pages(5, 31000114, 0, 2323242432)

Example 5 Use one of the following three methods to calculate space in a database with reserved_pages:

• Use case expressions to select a value appropriate for the index you are inspecting, selecting all non-log indexes in sysindexes for this database. In this query:
The data has a value of “index 0”, and is available when you include the statements when sysindexes.indid = 0 or sysindexes.indid = 1.

indid values greater than 1 for are indexes. Because this query does not sum the data space into the index count, it does not include a page count for indid of 0.

Each object has an index entry for index of 0 or 1, never both.

This query counts index 0 exactly once per table.

```sql
select 'data rsvd' = sum( case
    when indid > 1 then 0
    else reserved_pages(db_id(), id, 0)
end ),
'index rsvd' = sum( case
    when indid = 0 then 0
    else reserved_pages(db_id(), id, indid)
end )
from sysindexes
where id != 8

<table>
<thead>
<tr>
<th>data rsvd</th>
<th>index rsvd</th>
</tr>
</thead>
<tbody>
<tr>
<td>812</td>
<td>1044</td>
</tr>
</tbody>
</table>
```

Query sysindexes multiple times to display results after all queries are complete:

```sql
declare @data int,
@dbsize int,
@dataused int,
@indices int,
@indused int
select @data = sum( reserved_pages(db_id(), id, 0) ),
       @dataused = sum( used_pages(db_id(), id, 0) )
from sysindexes
where id != 8
and indid <= 1

select @indices = sum( reserved_pages(db_id(), id, indid) ),
       @indused = sum( used_pages(db_id(), id, indid) )
from sysindexes
where id != 8 and indid > 0

select @dbsize as 'db size',
@data as 'data rsvd'
<table>
<thead>
<tr>
<th>db size</th>
<th>data rsvd</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>820</td>
</tr>
</tbody>
</table>
```
• Query sysobjects for data space information and sysindexes for index information. From sysobjects, select table objects: [S]ystem or [U]ser:

```sql
declare @data int,
       @dbsize int,
       @dataused int,
       @indices int,
       @indused int
select @data = sum( reserved_pages(db_id(), id, 0) ),
       @dataused = sum( used_pages(db_id(), id, 0) )
from sysobjects
where id != 8
and type in ('S', 'U')
select @indices = sum( reserved_pages(db_id(), id, indid) ),
       @indused = sum( used_pages(db_id(), id, indid) )
from sysindexes
where id != 8
and indid > 0
select @dbsize as 'db size',
       @data as 'data rsvd',
       @dataused as 'data used',
       @indices as 'index rsvd',
       @indused as 'index used'
```

### Usage

- If a clustered index exists on an all-pages locked table, passing an index ID of 0 reports the reserved data pages, and passing an index ID of 1 reports the reserved index pages. All erroneous conditions result in a value of zero being returned.

- `reserved_pages` counts whatever you specify: if you supply a valid database, object, index (data is “index 0” for every table), it returns the reserved space for this database, object, or index. However, it can also count a database, object, or index multiple times. If you have it count the data space for every index in a table with multiple indexes, you get it counts the data space once for every index. If you sum these results, you get the number of indexes multiplied by the total data space, not the total number of data pages in the object.

- For Adaptive Server version 15.0, `reserved_pages` replaces the `reserved_pgs` function. These are the differences between `reserved_pages` and `reserved_pgs`. 
In Adaptive Server versions 12.5 and earlier, Adaptive Server stored OAM pages for the data and index in `sysindexes`. In Adaptive Server versions 15.0 and later, this information is stored per-partition in `sysparitions`. Because this information is stored differently, `reserved_pages` and `reserved_pgs` require different parameters and have different result sets.

`reserved_pgs` required a page ID. If you supplied a value that did not have a matching `sysindexes` row, the supplied page ID was 0 (for example, the data OAM page of a nonclustered index row). Because 0 was never a valid OAM page, if you supplied a page ID of 0, `reserved_pgs` returned 0; because the input value is invalid, `reserved_pgs` could not count anything.

However, `reserved_pages` requires an index ID, and 0 is a valid index ID (for example, data is “index 0” for every table). Because `reserved_pages` can not tell from the context that you do not require it to recount the data space for any index row except indid 0 or 1, it counts the data space every time you pass 0 as an index ID. Because `reserved_pages` counts this data space once per row, its yields a sum many times the true value.

These differences are described as:

- `reserved_pgs` does not affect the sum if you supply 0 as a value for the page ID for the OAM page input; it just returns a value of 0.

- If you supply `reserved_pages` with a value of 0 as the index ID, it counts the data space. Issue `reserved_pages` only when you want to count the data or you will affect the sum.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute `reserved_pgs`.

**See also**
- **Command** `update statistics`
- **Function** `data_pages`, `reserved_pages`, `row_count`, `used_pages`
reverse

Description
Returns the specified string with characters listed in reverse order.

Syntax
reverse(expression | uchar_expr)

Parameters
expression
is a character or binary-type column name, variable, or constant expression of char, varchar, nchar, nvarchar, binary, or varbinary type.

uchar_expr
is a character or binary-type column name, variable, or constant expression of unichar or univarchar type.

Examples
Example 1
select reverse("abcd")
-----
dcba

Example 2
select reverse(0x12345000)
--------
0x00503412

Usage

- reverse, a string function, returns the reverse of expression.
- If expression is NULL, reverse returns NULL.
- Surrogate pairs are treated as indivisible and are not reversed.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute reverse.

See also
For general information about string functions, see “String functions” on page 76.

Functions
lower, upper
right

The rightmost part of the expression with the specified number of characters.

Syntax

right(expression, integer_expr)

Parameters

expression
is a character or binary-type column name, variable, or constant expression of char, varchar, nchar, unichar, nvarchar, univarchar, binary, or varbinary type.

integer_expr
is any integer (tinyint, smallint, or int) column name, variable, or constant expression.

Examples

Example 1

```sql
select right("abcde", 3)
```

```sql
---
cde
```

Example 2

```sql
select right("abcde", 2)
```

```sql
--
de
```

Example 3

```sql
select right("abcde", 6)
```

```sql
-----
abcde
```

Example 4

```sql
select right(0x12345000, 3)
```

```sql
-------
0x345000
```

Example 5

```sql
select right(0x12345000, 2)
```

```sql
------
0x5000
```

Example 6

```sql
select right(0x12345000, 6)
```

```sql
--------
0x12345000
```
right

Usage
- right, a string function, returns the specified number of characters from the rightmost part of the character or binary expression.
- If the specified rightmost part begins with the second surrogate of a pair (the low surrogate), the return value starts with the next full character. Therefore, one less character is returned.
- The return value has the same datatype as the character or binary expression.
- If expression is NULL, right returns NULL.

Standards
ANSI SQL – Compliance level: Transact-SQL extension

Permissions
Any user can execute right.

See also
For general information about string functions, see “String functions” on page 76.

Functions
rtrim, substring
**rm_appcontext**

**Description**
Removes a specific application context, or all application contexts. `rm_appcontext` is a function provided by the Application Context Facility (ACF).

**Syntax**
```
rm_appcontext("context_name", "attribute_name")
```

**Parameters**
- `context_name` is a row specifying an application context name. It is saved as datatype `char(30)`.
- `attribute_name` is a row specifying an application context attribute name. It is saved as datatype `char(30)`.

**Examples**
**Example 1** Removes an application context by specifying some or all attributes:
```
select rm_appcontext("CONTEXT1", "*")
----------
0

select rm_appcontext("\", "\")
----------
0

select rm_appcontext("NON_EXISTING_CTX", "ATTR")
----------
-1
```

**Example 2** Shows the result when a user without appropriate permissions attempts to remove an application context:
```
select rm_appcontext("CONTEXT1", "ATTR2")
----------
-1
```

**Usage**
- This function always returns 0 for success.
- All the arguments for this function are required.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Permissions depend on the user profile and the application profile, which are stored by ACF.

**See also**
For more information on the ACF see “Row-level access control” in Chapter 11, “Managing User Permissions” of the *System Administration Guide*.

**Functions**
- `get_appcontext`
- `list_appcontext`
- `set_appcontext`
role_contain

Description
Returns 1 if role2 contains role1.

Syntax
role_contain("role1", "role2")

Parameters
role1
is the name of a system or user-defined role.

role2
is the name of another system or user-defined role.

Examples

Example 1
select role_contain("intern_role", "doctor_role")
-----------
1

Example 2
select role_contain("specialist_role", "intern_role")
-----------
0

Usage
role_contain, a system function, returns 1 if role1 is contained by role2.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute role_contain.

See also
For more information about system functions, see “System functions” on page 77.

Documents
For more information about contained roles and role hierarchies, see the System Administration Guide.

Functions
mut_excl_roles, proc_role, role_id, role_name

Commands
alter role

System procedures
sp_activeroles, sp_displayroles, sp_role
**role_id**

**Description**
Returns the system role ID of the name you specify.

**Syntax**

```
role_id("role_name")
```

**Parameters**

`role_name` is the name of a system or user-defined role. Role names and role IDs are stored in the `syssrvroles` system table.

**Examples**

**Example 1** Returns the system role ID of `sa_role`:

```
select role_id("sa_role")
------
0
```

**Example 2** Returns the system role ID of the “`intern_role`”:

```
select role_id("intern_role")
------
6
```

**Usage**

- `role_id`, a system function, returns the system role ID (`srid`). System role IDs are stored in the `srid` column of the `syssrvroles` system table.
- If the `role_name` is not a valid role in the system, Adaptive Server returns `NULL`.

**Standards**

ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**

Any user can execute `role_id`.

**See also**

For more information about system functions, see “System functions” on page 77.

**Documents**

For more information about roles, see the *System Administration Guide*.

**Functions**

`mut_excl_roles`, `proc_role`, `role_contain`, `role_name`
**role_name**

**Description**
Returns the name of a system role ID you specify.

**Syntax**
role_name(role_id)

**Parameters**
- **role_id**
  is the system role ID (sr_id) of the role. Role names are stored in syssrvroles.

**Examples**
```
select role_name(01)
```

```
-----------------------------
ss_role
```

**Usage**
role_name, a system function, returns the role name.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension

**Permissions**
Any user can execute role_name.

**See also**
For more information about system functions, see “System functions” on page 77.

**Functions**
mut_excl_roles, proc_role, role_contain, role_id
round

Description
Returns the value of the specified number, rounded to a specified number of decimal places.

Syntax
\[ \text{round}(\text{number}, \text{decimal\_places}) \]

Parameters
- **number**
  is any exact numeric (numeric, dec, decimal, tinyint, smallint, int, or bigint), approximate numeric (float, real, or double precision), or money column, variable, constant expression, or a combination of these.
- **decimal\_places**
  is the number of decimal places to round to.

Examples

**Example 1**

```
select round(123.4545, 2)
------------------------
 123.4500
```

**Example 2**

```
select round(123.45, -2)
------------------------
 100.00
```

**Example 3**

```
select round(1.2345E2, 2)
------------------------
 123.450000
```

**Example 4**

```
select round(1.2345E2, -2)
------------------------
 100.000000
```

Usage
- **round**, a mathematical function, rounds the **number** so that it has **decimal\_places** significant digits.
- A positive value for **decimal\_places** determines the number of significant digits to the right of the decimal point; a negative value for **decimal\_places** determines the number of significant digits to the left of the decimal point.
- Results are of the same type as **number** and, for numeric and decimal expressions, have an internal precision equal to the precision of the first argument plus 1 and a scale equal to that of **number**.
round always returns a value. If `decimal_places` is negative and exceeds the number of significant digits specified for `number`, Adaptive Server returns 0. (This is expressed in the form 0.00, where the number of zeros to the right of the decimal point is equal to the scale of `numeric`.) For example, the following returns a value of 0.00:

```sql
select round(55.55, -3)
```

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute `round`.

**See also**
For general information about mathematical functions, see “Mathematical functions” on page 74.

**Functions**
abs, ceiling, floor, sign, str
row_count

Description
Returns an estimate of the number of rows in the specified table.

Syntax
row_count(dbid, object_id [, ptnid])

Parameters
- `dbid` is the database ID where target object resides
- `object_id` is the object ID of table
- `ptnid` is the partition ID of interest

Examples
**Example 1** Returns an estimate of the number of rows in the given object:
```sql
select row_count(5, 31000114)
```

**Example 2** Returns an estimate of the number of rows in the specified partition (with partition ID of 2323242432) of the object with object ID of 31000114:
```sql
select row_count(5, 31000114, 2323242432)
```

Usage
All erroneous conditions will return in a value of zero being returned.

Standards
ANSI SQL – Compliance level: Transact-SQL extension

Permissions
Any user can execute `row_count`.

See also
- **Functions** reserved_pages, used_pages
**rtrim**

**Description**
Returns the specified expression, trimmed of trailing blanks.

**Syntax**
rtrim(char_expr | uchar_expr)

**Parameters**
- **char_expr**
  is a character-type column name, variable, or constant expression of char, varchar, nchar, or nvarchar type.

- **uchar_expr**
  is a character-type column name, variable, or constant expression of unichar or univarchar type.

**Examples**

```sql
select rtrim("abcd   ")
--------
abcd
```

**Usage**
- `rtrim`, a string function, removes trailing blanks.
- For Unicode, a blank is defined as the Unicode value U+0020.
- If `char_expr` or `uchar_expr` is NULL, returns NULL.
- Only values equivalent to the space character in the current character set are removed.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute `rtrim`.

**See also**
For general information about string functions, see “String functions” on page 76.

**Function**
- `ltrim`
sdc_intempdbconfig

Description: Cluster environments only – returns 1 if the system is currently in temporary database configuration mode; if not, returns 0.

Syntax: `sdc_intempdbconfig()`

Examples: `select sdc_intempdbconfig()`

Standards: ANSI SQL – Compliance level: Transact-SQL extension.

Permissions: Any user can run `sdc_intempdbconfig`. 

---

Reference Manual: Building Blocks 265
**set_appcontext**

**Description**
Sets an application context name, attribute name, and attribute value for a user session, defined by the attributes of a specified application. `set_appcontext` is a built-in function that the Application Context Facility (ACF) provides.

**Syntax**
```
set_appcontext("context_name", "attribute_name", "attribute_value")
```

**Parameters**
- **context_name**
  is a row that specifies an application context name. It is saved as the datatype `char(30)`.
- **attribute_name**
  is a row that specifies an application context attribute name. It is saved as the datatype `char(30)`.
- **attribute_value**
  is a row that specifies and application attribute value. It is saved as the datatype `char(30)`.

**Examples**

**Example 1** Creates an application context called CONTEXT1, with an attribute ATTR1 that has the value VALUE1.
```
select set_appcontext ("CONTEXT1", "ATTR1", "VALUE1")
---------------------
0
```

Attempting to override the existing application context created causes:
```
select set_appcontext ("CONTEXT1", "ATTR1", "VALUE1")
---------------------
-1
```

**Example 2** Shows `set_appcontext` including a datatype conversion in the value.
```
declare@numericvarchar varchar(25)
select @numericvarchar = "20"
select set_appcontext ("CONTEXT1", "ATTR2",
        convert(char(20), @numericvarchar))
---------------------
0
```

**Example 3** Shows the result when a user without appropriate permissions attempts to set the application context.
```
select set_appcontext ("CONTEXT1", "ATTR2", "VALUE1")
---------------------
-1
```
Usage

- set_appcontext returns 0 for success and -1 for failure.
- If you set values that already exist in the current session, set_appcontext returns -1.
- This function cannot override the values of an existing application context. To assign new values to a context, remove the context and re-create it using new values.
- set_appcontext saves attributes as char datatypes. If you are creating an access rule that must compare the attribute value to another datatype, the rule should convert the char data to the appropriate datatype.
- All the arguments for this function are required.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Permissions depend on the user profile and the application profile, stored by ACF.

See also

Documents For more information on the ACF see “Row-level access control” in Chapter 11, “Managing User Permissions” of the System Administration Guide.

Functions get_appcontext, list_appcontext, rm_appcontext
show_role

Description
Shows the login’s currently active system-defined roles.

Syntax
show_role()

Parameters
None.

Examples
Example 1

```
select show_role()
  sa_role sso_role oper_role replication_role
```

Example 2

```
if charindex("sa_role", show_role()) >0
begin
  print "You have sa_role"
end
```

Usage
- show_role, a system function, returns the login’s current active system-defined roles, if any (sa_role, sso_role, oper_role, or replication_role). If the login has no roles, show_role returns NULL.
- When a Database Owner invokes show_role after using setuser, show_role displays the active roles of the Database Owner, not the user impersonated with setuser.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute show_role.

See also
For general information about system functions, see “System functions” on page 77.

Commands
alter role, create role, drop role, grant, set, revoke

Functions
proc_role, role_contain

System procedures
sp_activeroles, sp_displayroles, sp_role
**show_sec_services**

**Description**
Lists the security services that are active for the session.

**Syntax**
`show_sec_services()`

**Parameters**
None.

**Examples**
Shows that the user’s current session is encrypting data and performing replay detection checks:

```sql
select show_sec_services()
encryption, replay_detection
```

**Usage**
- Use `show_sec_services` to list the security services that are active during the session.
- If no security services are active, `show_sec_services` returns NULL.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute `show_sec_services`.

**See also**
*Functions* `is_sec_service_on`
**sign**

**Description**
Returns the sign (1 for positive, 0, or -1 for negative) of the specified value.

**Syntax**
```
sign(numeric)
```

**Parameters**
`numeric` is any exact numeric (numeric, dec, decimal, tinyint, smallint, int, or bigint), approximate numeric (float, real, or double precision), or money column, variable, constant expression, or a combination of these.

**Examples**

**Example 1**
```
select sign(-123)
-----------
  -1
```

**Example 2**
```
select sign(0)
---------
   0
```

**Example 3**
```
select sign(123)
---------
   1
```

**Usage**
- `sign`, a mathematical function, returns the positive (1), zero (0), or negative (-1).
- Results are of the same type, and have the same precision and scale, as the numeric expression.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute `sign`.

**See also**
For general information about mathematical functions, see “Mathematical functions” on page 74.

**Functions**
abs, ceiling, floor, round
**sin**

**Description**
Returns the sine of the specified angle (in radians).

**Syntax**
```
sin(approx_numeric)
```

**Parameters**
- `approx_numeric` is any approximate numeric (float, real, or double precision) column name, variable, or constant expression.

**Examples**
```
select sin(45)
```
```
-------------------
  0.850904
```

**Usage**
`sin`, a mathematical function, returns the sine of the specified angle (measured in radians).

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute `sin`.

**See also**
For general information about mathematical functions, see “Mathematical functions” on page 74.

**Functions**
- `cos`
- `degrees`
- `radians`
sortkey

Description
Generates values that can be used to order results based on collation behavior, which allows you to work with character collation behaviors beyond the default set of Latin character-based dictionary sort orders and case- or accent-sensitivity.

Syntax
sortkey(char_expression | uchar_expression[, (collation_name | collation_ID)])

Parameters
char_expression
is a character-type column name, variable, or constant expression of char, varchar, nchar, or nvarchar type.

uchar_expression
is a character-type column name, variable, or constant expression of unichar or univarchar type.

collation_name
is a quoted string or a character variable that specifies the collation to use. Table 2-11 on page 275 shows the valid values.

collation_ID
is an integer constant or a variable that specifies the collation to use. Table 2-11 on page 275 shows the valid values.

Examples
Example 1 Shows sorting by European language dictionary order:
select * from cust_table where cust_name like "TI%" order by (sortkey(cust_name, "dict")

Example 2 Shows sorting by simplified Chinese phonetic order:
select * from cust_table where cust_name like "TI%" order by (sortkey(cust-name, "gbpinyin")

Example 3 Shows sorting by European language dictionary order using the in-line option:
select * from cust_table where cust_name like "TI%" order by cust_french_sort

Example 4 Shows sorting by Simplified Chinese phonetic order using preexisting keys:
select * from cust_table where cust_name like "TI%" order by cust_chinese_sort.
sortkey, a system function, generates values that can be used to order results based on collation behavior. This allows you to work with character collation behaviors beyond the default set of Latin-character-based dictionary sort orders and case- or accent-sensitivity. The return value is a varbinary datatype value that contains coded collation information for the input string that is returned from the sortkey function.

For example, you can store the values returned by sortkey in a column with the source character string. To retrieve the character data in the desired order, include in the select statement an order by clause on the columns that contain the results of running sortkey.

sortkey guarantees that the values it returns for a given set of collation criteria work for the binary comparisons that are performed on varbinary datatypes.

- sortkey can generate up to six bytes of collation information for each input character. Therefore, the result from using sortkey may exceed the length limit of the varbinary datatype. If this happens, the result is truncated to fit. Since this limit is dependent on the logical page size of your server, truncation removes result bytes for each input character until the result string is less than the following for DOL and APL tables:

<table>
<thead>
<tr>
<th>Locking scheme</th>
<th>Page size</th>
<th>Maximum row length</th>
<th>Maximum column length</th>
</tr>
</thead>
<tbody>
<tr>
<td>APL tables</td>
<td>2K (2048 bytes)</td>
<td>1962</td>
<td>1960 bytes</td>
</tr>
<tr>
<td></td>
<td>4K (4096 bytes)</td>
<td>4010</td>
<td>4008 bytes</td>
</tr>
<tr>
<td></td>
<td>8K (8192 bytes)</td>
<td>8106</td>
<td>8104 bytes</td>
</tr>
<tr>
<td></td>
<td>16K (16384 bytes)</td>
<td>16298</td>
<td>16296 bytes</td>
</tr>
<tr>
<td>DOL tables</td>
<td>2K (2048 bytes)</td>
<td>1964</td>
<td>1958 bytes</td>
</tr>
<tr>
<td></td>
<td>4K (4096 bytes)</td>
<td>4012</td>
<td>4006 bytes</td>
</tr>
<tr>
<td></td>
<td>8K (8192 bytes)</td>
<td>8108</td>
<td>8102 bytes</td>
</tr>
<tr>
<td></td>
<td>16K (16384 bytes)</td>
<td>16300</td>
<td>16294 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If table does not include any variable length columns</td>
</tr>
<tr>
<td></td>
<td>16K (16384 bytes)</td>
<td>16300</td>
<td>8191-6-2 = 8183 bytes</td>
</tr>
<tr>
<td></td>
<td>(subject to a max start offset of varlen = 8191)</td>
<td></td>
<td>If table includes at least one variable length column.*</td>
</tr>
</tbody>
</table>

* This size includes six bytes for the row overhead and two bytes for the row length field.

If this occurs, Adaptive Server issues a warning message, but the query or transaction that contained the sortkey function continues to run.
sortkey

- **char_expression** or **uchar_expression** must be composed of characters that are encoded in the server’s default character set.

- **char_expression** or **uchar_expression** can be an empty string. If it is an empty string, sortkey returns a zero-length varbinary value, and stores a blank for the empty string.

An empty string has a different collation value than an NULL string from a database column.

- If **char_expression** or **uchar_expression** is NULL, sortkey returns a null value.

- If a unicode expression has no specified sort order, Adaptive Server uses the binary sort order.

- If you do not specify a value for **collation_name** or **collation_ID**, sortkey assumes binary collation.

- The binary values generated from the sortkey function can change from one major version to another major version of Adaptive Server, such as version 12.0 to 12.5, version 12.9.2 to 12.0, and so on. If you are upgrading to the current version of Adaptive Server, regenerate keys and repopulate the shadow columns before any binary comparison takes place.

**Note** Upgrades from version 12.5 to 12.5.0.1 do not require this step, and Adaptive Server does not generate any errors or warning messages if you do not regenerate the keys. Although a query involving the shadow columns should work fine, the comparison result may differ from the pre-upgrade server.

Collation tables

There are two types of collation tables you can use to perform multilingual sorting:

1. A “built-in” collation table created by the sortkey function. This function exists in versions of Adaptive Server later than 11.5.1. You can use either the collation name or the collation ID to specify a built-in table.

2. An external collation table that uses the Unilib library sorting functions. You must use the collation name to specify an external table. These files are located in $SYBASE/collate/unicode.
Both of these methods work equally well, but a “built-in” table is tied to an Adaptive Server database, while an external table is not. If you use an Adaptive Server database, a built-in table provides the best performance. Both methods can handle any mix of English, European, and Asian languages.

There are two ways to use sortkey:

1. In-line – this uses sortkey as part of the order by clause and is useful for retrofitting an existing application and minimizing the changes. However, this method generates sort keys on-the-fly, and therefore does not provide optimum performance on large data sets of more than 1000 records.

2. Pre-existing keys – this method calls sortkey whenever a new record requiring multilingual sorting is added to the table, such as a new customer name. Shadow columns (binary or varbinary type) must be set up in the database, preferably in the same table, one for each desired sort order such as French, Chinese, and so on. When a query requires output to be sorted, the order by clause uses one of the shadow columns. This method produces the best performance since keys are already generated and stored, and are quickly compared only on the basis of their binary values.

You can view a list of available collation rules. Print the list by executing either sp_helpsort, or by querying and selecting the name, id, and description from syscharsets (type is between 2003 and 2999).

• Table 2-11 lists the valid values for collation name and collation ID.

Table 2-11: Collation names and IDs

<table>
<thead>
<tr>
<th>Description</th>
<th>Collation name</th>
<th>Collation ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Unicode multilingual</td>
<td>default</td>
<td>20</td>
</tr>
<tr>
<td>Thai dictionary order</td>
<td>thaidict</td>
<td>21</td>
</tr>
<tr>
<td>ISO14651 standard</td>
<td>iso14651</td>
<td>22</td>
</tr>
<tr>
<td>UTF-16 ordering – matches UTF-8 binary ordering</td>
<td>utf8bin</td>
<td>24</td>
</tr>
<tr>
<td>CP 850 Alternative – no accent</td>
<td>altnoacc</td>
<td>39</td>
</tr>
<tr>
<td>CP 850 Alternative – lowercase first</td>
<td>altdict</td>
<td>45</td>
</tr>
<tr>
<td>CP 850 Western European – no case preference</td>
<td>altncsp</td>
<td>46</td>
</tr>
<tr>
<td>CP 850 Scandinavian – dictionary ordering</td>
<td>scandict</td>
<td>47</td>
</tr>
<tr>
<td>CP 850 Scandinavian – case-insensitive with preference</td>
<td>scanncsp</td>
<td>48</td>
</tr>
<tr>
<td>GB Pinyin</td>
<td>gbpinyin</td>
<td>n/a</td>
</tr>
<tr>
<td>Binary sort</td>
<td>binary</td>
<td>50</td>
</tr>
<tr>
<td>Latin-1 English, French, German dictionary</td>
<td>dict</td>
<td>51</td>
</tr>
<tr>
<td>Latin-1 English, French, German no case</td>
<td>nocase</td>
<td>52</td>
</tr>
</tbody>
</table>
### sortkey

<table>
<thead>
<tr>
<th>Description</th>
<th>Collation name</th>
<th>Collation ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latin-1 English, French, German no case, preference</td>
<td>nocasep</td>
<td>53</td>
</tr>
<tr>
<td>Latin-1 English, French, German no accent</td>
<td>noaccent</td>
<td>54</td>
</tr>
<tr>
<td>Latin-1 Spanish dictionary</td>
<td>espdict</td>
<td>55</td>
</tr>
<tr>
<td>Latin-1 Spanish no case</td>
<td>espnocs</td>
<td>56</td>
</tr>
<tr>
<td>Latin-1 Spanish no accent</td>
<td>espnoac</td>
<td>57</td>
</tr>
<tr>
<td>ISO 8859-5 Russian dictionary</td>
<td>rUSDICT</td>
<td>58</td>
</tr>
<tr>
<td>ISO 8859-5 Russian no case</td>
<td>rusnocs</td>
<td>59</td>
</tr>
<tr>
<td>ISO 8859-5 Cyrillic dictionary</td>
<td>cyrdict</td>
<td>63</td>
</tr>
<tr>
<td>ISO 8859-5 Cyrillic no case</td>
<td>cyrnocs</td>
<td>64</td>
</tr>
<tr>
<td>ISO 8859-7 Greek dictionary</td>
<td>elldict</td>
<td>65</td>
</tr>
<tr>
<td>ISO 8859-2 Hungarian dictionary</td>
<td>hundict</td>
<td>69</td>
</tr>
<tr>
<td>ISO 8859-2 Hungarian no accents</td>
<td>hunnoac</td>
<td>70</td>
</tr>
<tr>
<td>ISO 8859-2 Hungarian no case</td>
<td>hunnocs</td>
<td>71</td>
</tr>
<tr>
<td>ISO 8859-9 Turkish dictionary</td>
<td>turdict</td>
<td>72</td>
</tr>
<tr>
<td>ISO 8859-9 Turkish no accents</td>
<td>turknoac</td>
<td>73</td>
</tr>
<tr>
<td>ISO 8859-9 Turkish no case</td>
<td>turknocs</td>
<td>74</td>
</tr>
<tr>
<td>CP932 binary ordering</td>
<td>cp932bin</td>
<td>129</td>
</tr>
<tr>
<td>Chinese phonetic ordering</td>
<td>dynix</td>
<td>130</td>
</tr>
<tr>
<td>GB2312 binary ordering</td>
<td>gb2312bn</td>
<td>137</td>
</tr>
<tr>
<td>Common Cyrillic dictionary</td>
<td>cyrdict</td>
<td>140</td>
</tr>
<tr>
<td>Turkish dictionary</td>
<td>turdict</td>
<td>155</td>
</tr>
<tr>
<td>EUCKSC binary ordering</td>
<td>euckscbn</td>
<td>161</td>
</tr>
<tr>
<td>Chinese phonetic ordering</td>
<td>gbpinyin</td>
<td>163</td>
</tr>
<tr>
<td>Russian dictionary ordering</td>
<td>rUSDICT</td>
<td>165</td>
</tr>
<tr>
<td>SJIS binary ordering</td>
<td>sjisbin</td>
<td>179</td>
</tr>
<tr>
<td>EUCJIS binary ordering</td>
<td>eucjisbin</td>
<td>192</td>
</tr>
<tr>
<td>BIG5 binary ordering</td>
<td>big5bin</td>
<td>194</td>
</tr>
<tr>
<td>Shift-JIS binary order</td>
<td>sjisbin</td>
<td>259</td>
</tr>
</tbody>
</table>

#### Standards
ANSI SQL – Compliance level: Transact-SQL extension.

#### Permissions
Any user can execute `sortkey`.

#### See also
**Function**  compare

---

276 Adaptive Server Enterprise
**soundex**

**Description**
Returns a four-character code representing the way an expression sounds.

**Syntax**
soundex(char_expr | uchar_expr)

**Parameters**
- **char_expr** is a character-type column name, variable, or constant expression of char, varchar, nchar, or nvarchar type.
- **uchar_expr** is a character-type column name, variable, or constant expression of unichar or univarchar type.

**Examples**
```
select soundex ("smith"), soundex ("smythe")
```

```
S530  S530
```

**Usage**
- **soundex**, a string function, returns a four-character soundex code for character strings that are composed of a contiguous sequence of valid single- or double-byte roman letters.
- The **soundex** function converts an alphabetic string to a four-digit code for use in locating similar-sounding words or names. All vowels are ignored unless they constitute the first letter of the string.
- If **char_expr** or **uchar_expr** is NULL, returns NULL.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute soundex.

**See also**
For general information about string functions, see “String functions” on page 76.

Function difference
**space**

**Description**
Returns a string consisting of the specified number of single-byte spaces.

**Syntax**
```
space(integer_expr)
```

**Parameters**
- `integer_expr` is any integer (tinyint, smallint, or int) column name, variable, or constant expression.

**Examples**
```
select "aaa", space(4), "bbb"
--- ---- ---
aaa   bbb
```

**Usage**
`space`, a string function, returns a string with the indicated number of single-byte spaces.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute `space`.

**See also**
For general information about string functions, see “String functions” on page 76.

**Functions**
isnull, rtrim
### spid_instance_id

<table>
<thead>
<tr>
<th>Description</th>
<th>Cluster environments only – returns the instance ID on which the specified process id (spid) is running.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td><code>spid_instance_id(spid_value)</code></td>
</tr>
</tbody>
</table>
| Parameters  | **spid_value**  
the spid number whose instance id is requested |
| Examples    | Returns the ID of the instance that is running process id number 27:  
`select spid_instance_id(27)` |
| Usage       | • If you do not include a spid value, `spid_instance_id` returns NULL.  
• If you enter an invalid or non-existing process id value, `spid_instance_id` returns NULL. |
| Standards   | ANSI SQL – Compliance level: Transact-SQL extension. |
| Permissions | Any user can execute `spid_instance_id`. |
square

Description
Returns the square of a specified value expressed as a float.

Syntax
square(numeric_expression)

Parameters
numeric_expression
is a numeric expression of type float.

Examples
Example 1 Returns the square from an integer column:

    select square(total_sales) from titles

----------------
16769025.00000
15023376.00000
350513284.00000
...
16769025.00000
(18 row(s) affected)

Example 2 Returns the square from a money column:

    select square(price) from titles

----------
399.600100
142.802500
8.940100
NULL
...
224.700100
(18 row(s) affected)

Usage
This function is the equivalent of power(numeric_expression,2), but it returns type float rather than int.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute square.

See also
Function  power

Datatypes  exact_numeric, approximate_numeric, money, float
**sqrt**

**Description**
Returns the square root of the specified number.

**Syntax**
sqrt(approx_numeric)

**Parameters**
approx_numeric
is any approximate numeric (float, real, or double precision) column name, variable, or constant expression that evaluates to a positive number.

**Examples**
```
select sqrt(4)
2.000000
```

**Usage**
- sqrt, a mathematical function, returns the square root of the specified value.
- If you attempt to select the square root of a negative number, Adaptive Server returns the following error message:
  
  Domain error occurred.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute sqrt.

**See also**
For general information about mathematical functions, see “Mathematical functions” on page 74.

**Function**  power
**stddev**

**Description**
Computes the standard deviation of a sample consisting of a numeric expression, as a double.

**Note**  stddev and stdev are aliases for stddev_samp. See stddev_samp on page 287 for details.
**stdev**

Description

Computes the standard deviation of a sample consisting of a numeric expression, as a double.

**Note**  
stdev and std are aliases for stddev_samp. See stddev_samp on page 287 for details.
**stdevp**

**Description**

Computes the standard deviation of a population consisting of a numeric expression, as a double.

**Note**  
stdevp is an alias for stddev_pop. See stddev_pop on page 285 for details.
stddev_pop

Description  Computes the standard deviation of a population consisting of a numeric expression, as a double. stddevp is an alias for stddev_pop, and uses the same syntax.

Syntax  stddev_pop ( [ all | distinct ] expression )

Parameters  all
  applies stddev_pop to all values. all is the default.
distinct
  eliminates duplicate values before stddev_pop is applied.

expression  is the expression—commonly a column name—in which its population-based standard deviation is calculated over a set of rows.

Examples  The following statement lists the average and standard deviation of the advances for each type of book in the pubs2 database.

  select type, avg(advance) as "avg", stddev_pop(advance) as "stddev" from titles group by type order by type

Usage  Computes the population standard deviation of the provided value expression evaluated for each row of the group (if distinct was specified, then each row that remains after duplicates have been eliminated), defined as the square root of the population variance.

Figure 2-3: The formula for population-related statistical aggregate functions

\[ \sigma^2 = \frac{\sum (x_i - \mu)^2}{n} \]

\( \sigma^2 = \text{Variance} \)
\( n = \text{Population size} \)
\( \mu = \text{Mean of the values } x_i \)

Standards  ANSI SQL – Compliance level: Transact-SQL extension.

Permissions  Any user can execute stddev_pop.

See also  For general information about aggregate functions, see “Aggregate functions” on page 54.
stddev_pop

**Functions**  stddev_samp, var_pop, var_samp
**stddev_samp**

**Description**
Computes the standard deviation of a sample consisting of a numeric expression, as a double. stddev and stddev are aliases for stddev_samp, and use the same syntax.

**Syntax**
```
stddev_samp ( [ all | distinct ] expression )
```

**Parameters**
- `all` applies `stddev_samp` to all values. `all` is the default.
- `distinct` eliminates duplicate values before `stddev_samp` is applied.

**expression**
is any numeric datatype (float, real, or double precision) expression.

**Examples**
The following statement lists the average and standard deviation of the advances for each type of book in the pubs2 database.

```
select type, avg(advance) as "avg",
    stddev_samp(advance) as "stddev"
from titles
where total_sales > 2000 group by type order by type
```

**Usage**
Computes the sample standard deviation of the provided value expression evaluated for each row of the group (if `distinct` was specified, then each row that remains after duplicates have been eliminated), defined as the square root of the sample variance.

**Figure 2-4: The formula for sample-related statistical aggregate functions**

The formula that defines an unbiased estimate of the population variance from a sample of size $n$ having mean $\bar{x}$ (var_samp) is as follows. The sample standard deviation (stddev_samp) is the positive square root of this.

$$s^2 = \frac{\sum (x_i - \bar{x})^2}{n - 1}$$

- $s^2 = $ Variance
- $n = $ Sample size
- $\bar{x} = $ Mean of the values $x_i$

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute `stddev_samp`.

**See also**
For general information about aggregate functions, see “Aggregate functions” on page 54.
`stddev_samp`

**Functions**  `stddev_pop`, `var_pop`, `var_samp`
str

Description
Returns the character equivalent of the specified number.

Syntax
str(approx_numeric[, length[, decimal]])

Parameters
approx_numeric
is any approximate numeric (float, real, or double precision) column name, variable, or constant expression.

length
sets the number of characters to be returned (including the decimal point, all digits to the right and left of the decimal point, and blanks). The default is 10.

decimal
sets the number of decimal digits to be returned. The default is 0.

Examples
Example 1
select str(1234.7, 4)
-----
1235

Example 2
select str(-12345, 6)
-------
-12345

Example 3
select str(123.45, 5, 2)
-----
123.5

Usage
• str, a string function, returns a character representation of the floating point number. For general information about string functions, see “String functions” on page 76.

• length and decimal are optional. If given, they must be nonnegative. str rounds the decimal portion of the number so that the results fit within the specified length. The length should be long enough to accommodate the decimal point and, if negative, the number’s sign. The decimal portion of the result is rounded to fit within the specified length. If the integer portion of the number does not fit within the length, however, str returns a row of asterisks of the specified length. For example:
  --
**

```
select str(123.456, 2, 4)
```

A short `approx_numeric` is right-justified in the specified length, and a long `approx_numeric` is truncated to the specified number of decimal places.

- If `approx_numeric` is NULL, returns NULL.

**Standards**

ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**

Any user can execute `str`.

**See also**

*Functions*  
abs, ceiling, floor, round, sign
**str_replace**

**Description**

Replaces any instances of the second string expression (\textit{string_expression2}) that occur within the first string expression (\textit{string_expression1}) with a third expression (\textit{string_expression3}).

**Syntax**

\[
\text{str_replace}(\textit{string_expression1}, \textit{string_expression2}, \textit{string_expression3})
\]

**Parameters**

\textit{string_expression1}

is the source string, or the string expression to be searched, expressed as char, varchar, unichar, univarchar, varbinary, or binary datatype.

\textit{string_expression2}

is the pattern string, or the string expression to find within the first expression (\textit{string_expression1}). \textit{string_expression2} is expressed as char, varchar, unichar, univarchar, varbinary, or binary datatype.

\textit{string_expression3}

is the replacement string expression, expressed as char, varchar, unichar, univarchar, binary, or varbinary datatype.

**Examples**

**Example 1** Replaces the string \textit{def} within the string \textit{cdefghi} with \textit{yyy}.

\[
\text{str_replace}("cdefghi","def","yyy")
\]

\[
\text{cyygh}\text{i}
\]

(1 row(s) affected)

**Example 2** Replaces all spaces with \textit{toyota}.

\[
\text{select str_replace("chevy, ford, mercedes", "", "toyota")}
\]

\[
\text{chevy,toyotaford,toyotamercedes}
\]

(1 row(s) affected)

**Note**

Adaptive Server converts an empty string constant to a string of one space automatically, to distinguish the string from NULL values.

**Example 3** Returns \textit{“abcghijklm”}:

\[
\text{select str_replace("abcdefghiklm", "def", NULL)}
\]

\[
\text{abcghijklm}
\]

(1 row affected)

**Usage**

- Returns varchar data if \textit{string_expression} (1, 2, or 3) is char or varchar.
• Returns univarchar data if \textit{string\_expression} (1, 2, or 3) is unichar or univarchar.

• Returns varbinary data if \textit{string\_expression} (1, 2, or 3) is binary or varbinary.

• All arguments must share the same datatype.

• If any of the three arguments is NULL, the function returns null.

\texttt{str\_replace} accepts NULL in the third parameter and treats it as an attempt to replace \textit{string\_expression}2 with NULL, effectively turning \texttt{str\_replace} into a “string cut” operation.

For example, the following returns “abcghijklm”:

\[
\texttt{str\_replace("abcdefghijklm", "def", NULL)}
\]

• The result length may vary, depending upon what is known about the argument values when the expression is compiled. If all arguments are variables with known constant values, Adaptive Server calculates the result length as:

\[
\text{result\_length} = ((s/p)\ast(r-p)+s)
\]

where

\begin{align*}
  s &= \text{length of source string} \\
  p &= \text{length of pattern string} \\
  r &= \text{length of replacement string}
\end{align*}

if \((r-p) \leq 0\), result length = \(s\)

• If the source string (\textit{string\_expression}1) is a column, and \textit{string\_expression}2 and \textit{string\_expression}3 are constant values known at compile time, Adaptive Server calculates the result length using the formula above.

• If Adaptive Server cannot calculate the result length because the argument values are unknown when the expression is compiled, the result length used is 255, unless traceflag 244 is on. In that case, the result length is 16384.

• \texttt{result\_len} never exceeds 16384.

\textbf{Standards}  
ANSI SQL – Compliance level: Transact-SQL extension.

\textbf{Permissions}  
Any user can execute \texttt{str\_replace}.

\textbf{See also}  
Datatypes  
char, varchar, binary, varbinary, unichar, univarchar

\textbf{Function}  
\texttt{length}
strtobin

Description
Converts a sequence of alphanumeric characters to their equivalent hexadecimal digits.

Syntax
```
select strtobin("string of valid alphanumeric characters")
```

Parameters
- `string of valid alphanumeric characters`
  - is string of valid alphanumeric characters, which consists of [1 – 9], [a – f] and [A – F].

Examples
**Example 1** Converts the alphanumeric string of “723ad82fe” to a sequence of hexadecimal digits:
```
select strtobin("723ad82fe")
go
```
```
0x0723ad82fe
```
The in-memory representation of the alphanumeric character string and its equivalent hexadecimal digits are:

<table>
<thead>
<tr>
<th>Alphanumeric character string (9 bytes)</th>
<th>Hexadecimal digits (5 bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 7 2 3  a  d  8 2 f  e</td>
<td>0 7 2 3  a  d  8 2 f  e</td>
</tr>
</tbody>
</table>

The function processes characters from right to left. In this example, the number of characters in the input is odd. For this reason, the hexadecimal sequence has a prefix of “0” and is reflected in the output.

**Example 2** Converts the alphanumeric string of a local variable called `@str_data` to a sequence of hexadecimal digits equivalent to the value of “723ad82fe”:
```
declare @str_data varchar(30)
select @str_data = "723ad82fe"
select strtobin(@str_data)
go
```
```
0x0723ad82fe
```

Usage
- Any invalid characters in the input results in NULL as the output.
- The input sequence of hexadecimal digits must have a prefix of “0x”.
- A NULL input results in NULL output.
<table>
<thead>
<tr>
<th><strong>strtof</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Standards</td>
</tr>
<tr>
<td>Permissions</td>
</tr>
<tr>
<td>See also</td>
</tr>
</tbody>
</table>
stuff

Description
Returns the string formed by deleting a specified number of characters from one string and replacing them with another string.

Syntax
stuff(char_expr1 | uchar_expr1, start, length, char_expr2 | uchar_expr2)

Parameters
char_expr1
is a character-type column name, variable, or constant expression of char, varchar, nchar, or nvarchar type.

uchar_expr1
is a character-type column name, variable, or constant expression of unichar or univarchar type.

start
specifies the character position at which to begin deleting characters.

length
specifies the number of characters to delete.

char_expr2
is another character-type column name, variable, or constant expression of char, varchar, nchar, or nvarchar type.

uchar_expr2
is another character-type column name, variable, or constant expression of unichar or univarchar type.

Examples
Example 1
select stuff("abc", 2, 3, "xyz")
----
axyz

Example 2
select stuff("abcdef", 2, 3, null)
go
---
aef

Example 3
select stuff("abcdef", 2, 3, ")
----
a ef
Usage

- **stuff**, a string function, deletes *length* characters from *char_expr1* or *uchar_expr1* at *start*, then inserts *char_expr2* or *uchar_expr2* into *char_expr1* or *uchar_expr1* at *start*. For general information about string functions, see “String functions” on page 76.

- If the start position or the length is negative, a NULL string is returned. If the start position is zero or longer than *expr1*, a NULL string is returned. If the length to be deleted is longer than *expr1*, *expr1* is deleted through its last character (see Example 1).

- If the start position falls in the middle of a surrogate pair, start is adjusted to be one less. If the start length position falls in the middle of a surrogate pair, length is adjusted to be one less.

- To use stuff to delete a character, replace *expr2* with NULL rather than with empty quotation marks. Using ‘‘’ to specify a null character replaces it with a space (see Examples 2 and 3).

- If *char_expr1* or *uchar_expr1* is NULL, stuff returns NULL. If *char_expr1* or *uchar_expr1* is a string value and *char_expr2* or *uchar_expr2* is NULL, stuff replaces the deleted characters with nothing.

- If you give a varchar expression as one parameter and a unichar expression as the other, the varchar expression is implicitly converted to unichar (with possible truncation).

### Standards

ANSI SQL – Compliance level: Transact-SQL extension.

### Permissions

Any user can execute stuff.

### See also

- **Functions**  
  replicate, substring
CHAPTER 2  Transact-SQL Functions

substring

Description
Returns the string formed by extracting the specified number of characters from another string.

Syntax
substring(expression, start, length)

Parameters
expression
is a binary or character column name, variable, or constant expression. Can be char, nchar, unichar, varchar, univarchar, or nvarchar data, binary, or varbinary.

start
specifies the character position at which the substring begins.

length
specifies the number of characters in the substring.

Examples

Example 1 Displays the last name and first initial of each author, for example, “Bennet A.”:

select au_lname, substring(au_fname, 1, 1)
from authors

Example 2 Converts the author’s last name to uppercase, then displays the first three characters:

select substring(upper(au_lname), 1, 3)
from authors

Example 3 Concatenates pub_id and title_id, then displays the first six characters of the resulting string:

select substring((pub_id + title_id), 1, 6)
from titles

Example 4 Extracts the lower four digits from a binary field, where each position represents two binary digits:

select substring(xactid,5,2)
from syslogs

Usage
• substring, a string function, returns part of a character or binary string. For general information about string functions, see “String functions” on page 76.

• If substring’s second argument is NULL, the result is NULL. If substring’s first or third argument is NULL, the result is blank..
**substring**

- If the start position from the beginning of `uchar_expr1` falls in the middle of a surrogate pair, `start` is adjusted to one less. If the start length position from the beginning of `uchar_expr1` falls in the middle of a surrogate pair, `length` is adjusted to one less.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute `substring`.

**See also**
**Functions** `charindex`, `patindex`, `stuff`
**sum**

**Description**
Returns the total of the values.

**Syntax**
```
sum([all | distinct] expression)
```

**Parameters**
- `all`  
  applies sum to all values. all is the default.
- `distinct`  
  eliminates duplicate values before sum is applied. distinct is optional.

**expression**
- is a column name, constant, function, any combination of column names, constants, and functions connected by arithmetic or bitwise operators, or a subquery. With aggregates, an expression is usually a column name. For more information, see “Expressions” on page 359.

**Examples**

**Example 1** Calculates the average advance and the sum of total sales for all business books. Each of these aggregate functions produces a single summary value for all of the retrieved rows:

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

**Example 2** Used with a group by clause, the aggregate functions produce single values for each group, rather than for the entire table. This statement produces summary values for each type of book:

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

**Example 3** Groups the titles table by publishers, and includes only those groups of publishers who have paid more than $25,000 in total advances and whose books average more than $15 in price:

```
select pub_id, sum(advance), avg(price)
from titles
group by pub_id
having sum(advance) > $25000 and avg(price) > $15
```

**Usage**
- sum, an aggregate function, finds the sum of all the values in a column. sum can only be used on numeric (integer, floating point, or money) datatypes. Null values are ignored in calculating sums.
- For general information about aggregate functions, see “Aggregate functions” on page 54.
When you sum integer data, Adaptive Server treats the result as an int value, even if the datatype of the column is smallint or tinyint. When you sum bigint data, Adaptive Server treats the result as a bigint. To avoid overflow errors in DB-Library programs, declare all variables for results of averages or sums appropriately.

- You cannot use sum with the binary datatypes.
- This function defines only numeric types; use with Unicode expressions generates an error.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute sum.

See also
Commands: compute clause, group by and having clauses, select, where clause
Functions: count, max, min
**suser_id**

**Description**
Returns the server user’s ID number from the syslogins table.

**Syntax**
suser_id([server_user_name])

**Parameters**
- server_user_name
  - is an Adaptive Server login name.

**Examples**

**Example 1**

```sql
select suser_id()
```

```
1
```

**Example 2**

```sql
select suser_id("margaret")
```

```
5
```

**Usage**
- suser_id, a system function, returns the server user’s ID number from syslogins. For general information about system functions, see “System functions” on page 77.
- To find the user’s ID in a specific database from the sysusers table, use the user_id system function.
- If no server_user_name is supplied, suser_id returns the server ID of the current user.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute suser_id.

**See also**
- Functions suser_name, user_id
suser_name

Description
Returns the name of the current server user or the user whose server ID is specified.

Syntax
suser_name([server_user_id])

Parameters
server_user_id
is an Adaptive Server user ID.

Examples
Example 1
    select suser_name()
    ----------------------------
    sa

Example 2
    select suser_name(4)
    -----------------------------
    margaret

Usage
suser_name, a system function, returns the server user’s name. Server user IDs are stored in syslogins. If no server_user_id is supplied, suser_name returns the name of the current user.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute suser_name.

See also
For general information about system functions, see “System functions” on page 77.

Functions
suser_id, user_name
**syb_quit**

**Description**
Terminates the connection.

**Syntax**
syb_quit()

**Examples**
Terminates the connection in which the function is executed and returns an error message.

    select syb_quit()

--------------
CT-LIBRARY error:
    ct_results(): network packet layer: internal net library error: Net-Library operation terminated due to disconnect

**Usage**
You can use syb_quit to terminate a script if the isql preprocessor command exit causes an error.

**Permissions**
Any user can execute syb_quit.
**syb_sendmsg**

**Description**  
UNIX only  Sends a message to a User Datagram Protocol (UDP) port.

**Syntax**  
syb_sendmsg *ip_address, port_number, message*

**Parameters**  
- *ip_address*  
is the IP address of the machine where the UDP application is running.
- *port_number*  
is the port number of the UDP port.
- *message*  
is the message to send. It can be up to 255 characters in length.

**Examples**  
**Example 1** Sends the message “Hello” to port 3456 at IP address 120.10.20.5:

```
select syb_sendmsg("120.10.20.5", 3456, "Hello")
```

**Example 2** Reads the IP address and port number from a user table, and uses a variable for the message to be sent:

```
declare @msg varchar(255)
select @msg = "Message to send"
select syb_sendmsg (ip_address, portnum, @msg)
from sendports
where username = user_name()
```

**Usage**  
- To enable the use of UDP messaging, a System Security Officer must set the configuration parameter `allow sendmsg` to 1.
- No security checks are performed with `syb_sendmsg`. Sybase strongly recommends that you do not use `syb_sendmsg` to send sensitive information across the network. By enabling this functionality, the user accepts any security problems that result from its use.
- For a sample C program that creates a UDP port, see `sp_sendmsg`.

**Standards**  
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**  
Any user can execute `syb_sendmsg`.

**See also**  
- System procedure `sp_sendmsg`
sys_tempdbid

Description
Cluster environments only – returns the id of the effective local system temporary database of the specified instance. Returns the id of the effective local system temporary database of the current instance when instance_id is not specified.

Syntax
sys_tempdbid(instance_id)

Parameters
instance_id
ID of the instance.

Examples
Returns the effective local system temporary database id for the instance with an instance id of 3:

```sql
select sys_tempdbid(3)
```

Usage
If you do not specify an instance ID, sys_tempdbid returns the id of the effective local system temporary database for the current instance.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can run sys_tempdbid.
**tan**

**Description**

Returns the tangent of the specified angle (in radians).

**Syntax**

```
tan(angle)
```

**Parameters**

`angle` is the size of the angle in radians, expressed as a column name, variable, or expression of type `float`, `real`, `double precision`, or any datatype that can be implicitly converted to one of these types.

**Examples**

```
select tan(60)
```

```
---------------------
  0.320040
```

**Usage**

`tan`, a mathematical function, returns the tangent of the specified angle (measured in radians).

**Standards**

ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**

Any user can execute `tan`.

**See also**

For general information about mathematical functions, see “Mathematical functions” on page 74.

**Functions**

- `atan`
- `atn2`
- `degrees`
- `radians`
tempdb_id

Description
Reports the temporary database to which a given session is assigned. The input of the tempdb_id function is a server process ID, and its output is the temporary database to which the process is assigned. If you do not provide a server process, tempdb_id reports the dbid of the temporary database assigned to the current process.

Syntax
```
tempdb_id()
```

Examples
Finds all the server processes that are assigned to a given temporary database:
```
select spid from master..sysprocesses
where tempdb_id(spid) = db_id("tempdatabase")
```

Usage
```
select tempdb_id gives the same result as select @@tempdbid.
```

See also
Commands
```
select
```
**textptr**

**Description**
Returns a pointer to the first page of a text, image, or unitext column.

**Syntax**
```
textptr(column_name)
```

**Parameters**
- `column_name` is the name of a text column.

**Examples**

**Example 1** Uses the `textptr` function to locate the text column, `copy`, associated with `au_id` 486-29-1786 in the author’s `blurbs` table. The text pointer is placed in local variable `@val` and supplied as a parameter to the `readtext` command, which returns 5 bytes, starting at the second byte (offset of 1):

```
declare @val binary(16)
select @val = textptr(copy) from blurbs
where au_id = "486-29-1786"
readtext blurbs.copy @val 1 5
```

**Example 2** Selects the `title_id` column and the 16-byte text pointer of the `copy` column from the `blurbs` table:

```
select au_id, textptr(copy) from blurbs
```

**Usage**
- `textptr`, a text and image function, returns the text pointer value, a 16-byte `varbinary` value.
- If a text, unitext, or image column has not been initialized by a non-null insert or by any update statement, `textptr` returns a NULL pointer. Use `textvalid` to check whether a text pointer exists. You cannot use `writetext` or `readtext` without a valid text pointer.

**Note**
Trailing `f` in `varbinary` values are truncated when the values are stored in tables. If you are storing text pointer values in a table, use `binary` as the datatype for the column.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute `textptr`.

**See also**
For general information about text and image functions, see “Text and image functions” on page 78.

**Datatypes**
- text, image, and unitext datatypes

**Function**
- `textvalid`

**Commands**
- `insert`, `update`, `readtext`, `writetext`
textvalid

Description Returns 1 if the pointer to the specified text or unitext column is valid; 0 if it is not.

Syntax `textvalid("table_name.column_name", textpointer)`

Parameters `table_name.column_name` is the name of a table and its text column.

`textpointer` is a text pointer value.

Examples Reports whether a valid text pointer exists for each value in the blurb column of the texttest table:

```sql
select textvalid("texttest.blurb", textptr(blurb))
from texttest
```

Usage • textvalid, a text and image function, checks that a given text pointer is valid. Returns 1 if the pointer is valid, or 0 if it is not.

• The identifier for a text or an image column must include the table name.

• For general information about text and image functions, see “Text and image functions” on page 78.

Standards ANSI SQL – Compliance level: Transact-SQL extension.

Permissions Any user can execute textvalid.

See also • Datatypes text, image, and unitext datatypes

Function textptr
to_unichar

Description
Returns a unichar expression having the value of the integer expression.

Syntax

to_unichar(integer_expr)

Parameters

integer_expr
is any integer (tinyint, smallint, or int) column name, variable, or constant expression.

Usage

• to_unichar, a string function, converts a Unicode integer value to a Unicode character value.

• If a unichar expression refers to only half of a surrogate pair, an error message appears and the operation is aborted.

• If a integer_expr is NULL, to_unichar returns NULL.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute to_unichar.

See also
For general information about string functions, see “String functions” on page 76.

Datatypes text, image, and unitext datatypes

Function char
 tran_dumpable_status

Description
Returns a true/false indication of whether dump transaction is allowed.

Syntax
tran_dumpable_status("database_name")

Parameters
database_name
is the name of the target database.

Examples
Checks to see if the pubs2 database can be dumped:

```sql
1> select tran_dumpable_status("pubs2")
2> go

-----------
106
```

(1 row affected)

In this example, you cannot dump pubs2. The return code of 106 is a sum of all the conditions met (2, 8, 32, 64). See the Usage section for a description of the return codes.

Usage
tran_dumpable_status allows you to determine if dump transaction is allowed on a database without having to run the command. tran_dumpable_status performs all of the checks that Adaptive Server performs when dump transaction is issued.

If tran_dumpable_status returns 0, you can perform the dump transaction command on the database. If it returns any other value, it cannot. The non-0 values are:

- 1 – A database with the name you specified does not exist.
- 2 – A log does not exist on a separate device.
- 4 – The log first page is in the bounds of a data-only disk fragment.
- 8 – The trunc log on chkpt option is set for the database.
- 16 – Non-logged writes have occurred on the database.
- 32 – Truncate-only dump tran has interrupted any coherent sequence of dumps to dump devices.
- 64 – Database is newly created or upgraded. Transaction log may not be dumped until a dump database has been performed.
- 128 – Database durability does not allow transaction dumps.
- 256 – Database is read-only. dump transaction started a transaction, which is not allowed on read-only databases.
tran_dumpable_status

- 512 – Database is online for standby access. dump transaction started a transaction, which is not allowed on databases in standby access because the transaction would disturb the load sequence.
- 1024 – Database is an archive database, which do not support dump transaction.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute this function.

See also
Command dump transaction
tsequal
Description
Compares timestamp values to prevent update on a row that has been modified since it was selected for browsing.

Syntax
tsequal(browsed_row_timestamp, stored_row_timestamp)

Parameters
browsed_row_timestamp
is the timestamp column of the browsed row.

stored_row_timestamp
is the timestamp column of the stored row.

Examples
Retrieves the timestamp column from the current version of the publishers table and compares it to the value in the timestamp column that has been saved. If the values in the two timestamp columns are equal, tsequal updates the row. If the values are not equal, tsequal returns this error message:

update publishers
set city = "Springfield"
where pub_id = "0736"
and tsequal(timestamp, 0x0001000000002ea8)

Usage
• tsequal, a system function, compares the timestamp column values to prevent an update on a row that has been modified since it was selected for browsing. For general information about system functions, see “System functions” on page 77.

• tsequal allows you to use browse mode without calling the dbqual function in DB-Library. Browse mode supports the ability to perform updates while viewing data. It is used in front-end applications using Open Client and a host programming language. A table can be browsed if its rows have been timestamped.

• To browse a table in a front-end application, append the for browse keywords to the end of the select statement sent to Adaptive Server. For example:

Start of select statement in an Open Client application
... for browse

Completion of the Open Client application routine
• Do not use tsequal in the where clause of a select statement; only in the where clause of insert and update statements where the rest of the where clause matches a single unique row.
If you use a timestamp column as a search clause, compare it like a regular varbinary column; that is, timestamp1 = timestamp2.

Timestamping a new table for browsing

- When creating a new table for browsing, include a column named timestamp in the table definition. The column is automatically assigned a datatype of timestamp; you do not have to specify its datatype. For example:

```sql
create table newtable(col1 int, timestamp, col3 char(7))
```

Whenever you insert or update a row, Adaptive Server timestamps it by automatically assigning a unique varbinary value to the timestamp column.

Timestamping an existing table

- To prepare an existing table for browsing, add a column named timestamp using `alter table`. For example, to add a timestamp column with a NULL value to each existing row:

```sql
alter table oldtable add timestamp
```

To generate a timestamp, update each existing row without specifying new column values:

```sql
update oldtable
set col1 = col1
```

Standards

- ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

- Any user can execute `tsequal`.

See also

- **Datatype**  Timestamp datatype
uhighsurr

Description
Returns 1 if the Unicode value at position start is the high half of a surrogate pair (which should appear first in the pair). Returns 0 otherwise.

Syntax
uhighsurr(uchar_expr, start)

Parameters

- `uchar_expr` is a character-type column name, variable, or constant expression of unichar or univarchar type.
- `start` specifies the character position to investigate.

Usage
- `uhighsurr`, a string function, allows you to write explicit code for surrogate handling. Specifically, if a substring starts on a Unicode character where `uhighsurr` is true, extract a substring of at least 2 Unicode values (`substr` does not extract half of a surrogate pair).
- If `uchar_expr` is NULL, `uhighsurr` returns NULL.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute `uhighsurr`.

See also
For general information about string functions, see “String functions” on page 76.

Function
ulowsurr
ulowsurr

Description
Returns 1 if the Unicode value at position \textit{start} is the low half of a surrogate pair (which should appear second in the pair). Returns 0 otherwise.

Syntax
\texttt{ulowsurr(uchar\_expr, start)}

Parameters
\textit{uchar\_expr} is a character-type column name, variable, or constant expression of \texttt{unichar} or \texttt{univarchar} type.

\textit{start} specifies the character position to investigate.

Usage
- \texttt{ulowsurr}, a string function, allows you to write explicit code around adjustments performed by \texttt{substr}, \texttt{stuff}, and \texttt{right}. Specifically, if a substring ends on a Unicode value where \texttt{ulowsurr} is true, the user knows to extract a substring of 1 less characters (or 1 more). \texttt{substr} does not extract a string that contains an unmatched surrogate pair.

- If \texttt{uchar\_expr} is NULL, \texttt{ulowsurr} returns NULL.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute \texttt{ulowsurr}.

See also
For general information about string functions, see “String functions” on page 76.

Function \texttt{uhighsurr}
**upper**

Description
Returns the uppercase equivalent of the specified string.

Syntax
upper(char_expr)

Parameters
char_expr
is a character-type column name, variable, or constant expression of char, unichar, varchar, nchar, nvarchar, or univarchar type.

Examples
```
select upper("abcd")
```

----

ABCD

Usage

- upper, a string function, converts lowercase to uppercase, returning a character value.
- If char_expr or uchar_expr is NULL, upper returns NULL.
- Characters that have no upper-case equivalent are left unmodified.
- If a unichar expression is created containing only half of a surrogate pair, an error message appears and the operation is aborted.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute upper.

See also
For general information about string functions, see “String functions” on page 76.

**Function** lower
**uscalar**

Description Returns the Unicode scalar value for the first Unicode character in an expression.

Syntax `uscalar(uchar_expr)`

Parameters `uchar_expr` is a character-type column name, variable, or constant expression of unichar, or univarchar type.

Usage
- `uscalar`, a string function, returns the Unicode value for the first Unicode character in an expression.
- If `uchar_expr` is NULL, returns NULL.
- If `uscalar` is called on a `uchar_expr` containing an unmatched surrogate half, an error occurs and the operation is aborted.

Standards ANSI SQL – Compliance level: Transact-SQL extension.

Permissions Any user can execute `uscalar`.

See also For general information about string functions, see “String functions” on page 76.

**Functions** `ascii`
**used_pages**

**Description**
Reports the number of pages used by a table, an index, or a specific partition. Unlike `data_pages`, `used_pages` does include pages used for internal structures. This function replaces the `used_pgs` function used in versions of Adaptive Server earlier than 15.0.

**Syntax**
```
used_pages(dbid, object_id[, indid[, ptnid]])
```

**Parameters**
- `dbid` is the database id where target object resides.
- `object_id` is the object ID of the table for which you want to see the used pages. To see the pages used by an index, specify the object ID of the table to which the index belongs.
- `indid` is the index id of interest.
- `ptnid` is the partition id of interest.

**Examples**

**Example 1** Returns the number of pages used by the object with a object ID of 31000114 in the specified database (including any indexes):
```
select used_pages(5, 31000114)
```

**Example 2** Returns the number of pages used by the object in the data layer, regardless of whether or not a clustered index exists:
```
select used_pages(5, 31000114, 0)
```

**Example 3** Returns the number of pages used by the object in the index layer for an index with index ID 2. This does not include the pages used by the data layer (See the first bullet in the Usage section for an exception):
```
select used_pages(5, 31000114, 2)
```

**Example 4** Returns the number of pages used by the object in the data layer of the specific partition, which in this case is 232324232:
```
select used_pages(5, 31000114, 0, 232324232)
```

**Usage**
- In an all-pages locked table with a clustered index, the value of the last parameter determines which pages used are returned:
  - `used_pages(dbid, objid, 0)` – which explicitly passes 0 as the index ID, returns only the pages used by the data layer.
used_pages

- used_pages(dbid, objid, 1) – returns the pages used by the index layer as well as the pages used by the data layer.

To obtain the index layer used pages for an all-pages locked table with a clustered index, subtract used_pages(dbid, objid, 0) from used_pages(dbid, objid, 1).

- In an all-pages-locked table with a clustered index, used_pages is passed only the used pages in the data layer, for a value of indid = 0. When indid=1 is passed, the used pages at the data layer and at the clustered index layer are returned, as in previous versions.

- used_pages is similar to the old used_pgs(objid, doampg, ioampg) function.

- All erroneous conditions result in a return value of zero.

Standards  ANSI SQL – Compliance level: Transact-SQL extension.
Permissions Any user can execute used_pgs.
See also  Functions data_pages, object_id
user

Description
Returns the name of the current user.

Syntax
user

Parameters
None.

Examples

```
select user

------
dbo
```

Usage
- user, a system function, returns the user’s name.
- If the sa_role is active, you are automatically the Database Owner in any database you are using. Inside a database, the user name of the Database Owner is always “dbo”.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute user.

See also
For general information about system functions, see “System functions” on page 77.

Functions
user_name
**user_id**

**Description**
Returns the ID number of the specified user or of the current user in the database.

**Syntax**
user_id([user_name])

**Parameters**
- **user_name**
is the name of the user.

**Examples**

**Example 1**
```sql
select user_id()
```

```
--------
  1
```

**Example 2**
```sql
select user_id("margaret")
```

```
--------
   4
```

**Usage**
- user_id, a system function, returns the user’s ID number. For general information about system functions, see “System functions” on page 77.
- user_id reports the number from sysusers in the current database. If no user_name is supplied, user_id returns the ID of the current user. To find the server user ID, which is the same number in every database on Adaptive Server, use suser_id.
- Inside a database, the “guest” user ID is always 2.
- Inside a database, the user_id of the Database Owner is always 1. If you have the sa_role active, you are automatically the Database Owner in any database you are using. To return to your actual user ID, use set sa_role off before executing user_id. If you are not a valid user in the database, Adaptive Server returns an error when you use set sa_role off.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
You must System Administrator or System Security Officer to use this function on a user_name other than your own.

**See also**
- **Commands**
  - setuser
- **Functions**
  - suser_id, user_name
user_name

Description
Returns the name within the database of the specified user or of the current user.

Syntax
user_name([user_id])

Parameters
user_id
is the ID of a user.

Examples
Example 1
select user_name()

-----------------------------
dbo

Example 2
select user_name(4)

-----------------------------
margaret

Usage
• user_name, a system function, returns the user’s name, based on the user’s ID in the current database.
• If no user_id is supplied, user_name returns the name of the current user.
• If the sa_role is active, you are automatically the Database Owner in any database you are using. Inside a database, the user_name of the Database Owner is always “dbo”.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
You must be a System Administrator or System Security Officer to use this function on a user_id other than your own.

See also
For general information about system functions, see “System functions” on page 77.

Functions
suser_name, user_id
valid_name

Description
Returns 0 if the specified string is not a valid identifier or a number other than 0 if the string is a valid identifier, and can be up to 255 bytes in length.

Syntax
valid_name(character_expression[, maximum_length])

Parameters
character_expression
is a character-type column name, variable, or constant expression of char, varchar, nchar or nvarchar type. Constant expressions must be enclosed in quotation marks.

maximum_length
is an integer larger than 0 and less than or equal to 255. The default value is 30. If the identifier length is larger than the second argument, valid_name returns 0, and returns a value greater than zero if the identifier length is invalid.

Examples
Creates a procedure to verify that identifiers are valid:

```
create procedure chkname
@name varchar(30)
as
    if valid_name(@name) = 0
        print "name not valid"
```

Usage
• valid_name, a system function, returns 0 if the character_expression is not a valid identifier (illegal characters, more than 30 bytes long, or a reserved word), or a number other than 0 if it is a valid identifier.

• Adaptive Server identifiers can be a maximum of 16384 bytes in length, whether single-byte or multibyte characters are used. The first character of an identifier must be either an alphabetic character, as defined in the current character set, or the underscore (_) character. Temporary table names, which begin with the pound sign (#), and local variable names, which begin with the at sign (@), are exceptions to this rule. valid_name returns 0 for identifiers that begin with the pound sign (#) and the at sign (@).

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
Any user can execute valid_name.

See also
For general information about system functions, see “System functions” on page 77.

System procedure sp_checkreswords
valid_user

Description
Returns 1 if the specified ID is a valid user or alias in at least one database on this Adaptive Server.

Syntax
valid_user(server_user_id)

Parameters
server_user_id
is a server user ID. Server user IDs are stored in the suid column of syslogins.

Examples
select valid_user(4)
---------------
1

Usage
valid_user, a system function, returns 1 if the specified ID is a valid user or alias in at least one database on this Adaptive Server.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
You must be a System Administrator or a System Security Officer to use this function on a server_user_id other than your own.

See also
For general information about system functions, see “System functions” on page 77.

System procedures
sp_addlogin, sp_adduser
<table>
<thead>
<tr>
<th><strong>Description</strong></th>
<th>Computes the statistical variance of a sample consisting of a numeric expression, as a double, and returns the variance of a set of numbers.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Note</strong></td>
<td>var and variance are aliases of var_samp. See var_samp on page 329 for details.</td>
</tr>
</tbody>
</table>
**var_pop**

**Description**
Computes the statistical variance of a population consisting of a numeric expression, as a double. varp is an alias for var_pop, and uses the same syntax.

**Syntax**
```
var_pop ( [all | distinct] expression )
```

**Parameters**
- `all` applies var_pop to all values. all is the default.
- `distinct` eliminates duplicate values before var_pop is applied.

**expression**
is an expression—commonly a column name—in which its population-based variance is calculated over a set of rows.

**Examples**
Lists the average and variance of the advances for each type of book in the pubs2 database:
```
select type, avg(advance) as "avg", var_pop(advance) as "variance" from titles group by type order by type
```

**Usage**
Computes the population variance of the provided value expression evaluated for each row of the group (if distinct was specified, then each row that remains after duplicates have been eliminated), defined as the sum of squares of the difference of value expression, from the mean of value expression, divided by the number of rows in the group or partition.

**Figure 2-5: The formula for population-related statistical aggregate functions**

The formula that defines the variance of the population of size $n$ having mean $\mu$ (var_pop) is as follows. The population standard deviation (stddev_pop) is the positive square root of this.

$$
\sigma^2 = \frac{\sum (x_i - \mu)^2}{n}
\quad \sigma^2 = \text{Variance} \\
\quad n = \text{Population size} \\
\quad \mu = \text{Mean of the values } x_i
$$

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute var_pop.

**See also**
For general information about aggregate functions, see “Aggregate functions” in *Adaptive Server Enterprise Reference Manual: Building Blocks*. 
**var_pop**

**Functions**  stddev_pop, stddev_samp, var_samp
**var_samp**

**Description**
Computes the statistical variance of a sample consisting of a numeric-expression, as a `double`, and returns the variance of a set of numbers. `var` and `variance` are aliases of `var_samp`, and use the same syntax.

**Syntax**
```
var_samp ([ all | distinct] expression)
```

**Parameters**
- `all` applies `var_samp` to all values. all is the default.
- `distinct` eliminates duplicate values before `var_samp` is applied.
- `expression` is any numeric datatype (`float`, `real`, or `double`) expression.

**Examples**
Lists the average and variance of the advances for each type of book in the pubs2 database:
```
select type, avg(advance) as "avg", var_samp(advance) as "variance" from titles where total_sales > 2000 group by type order by type
```

**Usage**
`var_samp` returns a result of double-precision floating-point datatype. If applied to the empty set, the result is NULL.

*Figure 2-6: The formula for sample-related statistical aggregate functions*

The formula that defines an unbiased estimate of the population variance from a sample of size \( n \) having mean \( \bar{x} \) (\( \text{var}_\text{samp} \)) is as follows. The sample standard deviation (\( \text{stddev}_\text{samp} \)) is the positive square root of this.

\[
\begin{align*}
\text{Var}^2 &= \frac{\sum (x_i - \bar{x})^2}{n - 1} \\
\bar{x} &= \text{Mean of the values } x_i \\
n &= \text{Sample size} \\
\end{align*}
\]

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute `var_samp`.

**See also**
For general information about aggregate functions, see “Aggregate functions” in *Adaptive Server Enterprise Reference Manual: Building Blocks*.

**Functions**
`stddev_pop`, `stddev_samp`, `var_pop`


**variance**

**Description**

Computes the statistical variance of a sample consisting of a numeric expression, as a double, and returns the variance of a set of numbers.

**Note** var and variance are aliases of var_samp. See var_samp on page 329 for details.
**varp**

Description

Computes the statistical variance of a population consisting of a numeric expression, as a double.

---

**Note** varp is an alias of var_pop. See var_pop on page 327 for details.
workload_metric

Description
Cluster environments only – Queries the current workload metric for the instance you specify, or updates the metric for the instance you specify.

Syntax
workload_metric( instance_id | instance_name [, new_value ] )

Parameters
instance_id
ID of the instance.

instance_name
name of the instance.

new_value
float value representing the new metric.

Examples
Example 1 Sees the user metric on the current instance:

```sql
select workload_metric()
```

Example 2 Sees the user metric on instance “ase2”:

```sql
select workload_metric("ase2")
```

Example 3 Sets the value of the user metric on “ase3” to 27.54:

```sql
select workload_metric("ase3", 27.54)
```

Usage
- A NULL value indicates the current instance.
- If a value is specified for new_value, the specified value becomes the current user metric. If a value is not specified for new_value, the current workload metric is returned.
- The value of new_value must be zero or greater.
- If a value is supplied for new_value, workload_metric returns that value if the operation is successful. Otherwise, workload_metric returns -1.

Standards
ANSI SQL – Compliance level: Transact-SQL extension.

Permissions
You must have the sa_role or ha_role to execute workload_metric.
xa_bqual

Description
Returns the binary version of the bqual component of an ASCII XA transaction ID.

Syntax
xa_bqual(xid, 0)

Parameters
xid
is the ID of an Adaptive Server transaction, obtained from the xactname column in systransactions or from sp_transactions.

0
is reserved for future use

Examples
Example 1 Returns “0x227f06ca80”, the binary translation of the branch qualifier for the Adaptive Server transaction ID “0000000A_IphIT596iC7bF2#AUfkzaM_8DY6OE0”. The Adaptive Server transaction ID is first obtained using sp_transactions:

1> sp_transactions
2> go
...
--------------------------------------------------------------------
0x227f06ca80

Example 2 xa_bqual is often used together with xa_gtrid. This example returns the global transaction IDs and branch qualifiers from all rows in systransactions where its coordinator column is the value of “3”:

1> select gtrid=xa_gtrid(xactname,0),
   bqual=xa_bqual(xactname,0)
   from systransactions where coordinator = 3
2> go
   gtrid
**xa_bqual**

```
0xb1946cdc52464a61cba42fe4e0f5232b
0x227f06ca80
```

**Usage**

If an external transaction is blocked on Adaptive Server and you are using `sp_lock` and `sp_transactions` to identify the blocking transaction, you can use the XA transaction manager to terminate the global transaction. However, when you execute `sp_transactions`, the value of `xactname` it returns is in ASCII string format, while XA Server uses an undecoded binary value. Using `xa_bqual` thus allows you to determine the `bqual` portion of the transaction name in a format that can be understood by the XA transaction manager.

`xa_bqual` returns:

- The translated version of this string that follows the second “_” (underscore) and precedes either the third “_” or end-of-string value, whichever comes first.
- NULL if the transaction ID cannot be decoded, or is in an unexpected format.

**Note** `xa_bqual` does not perform a validation check on the xid, but only returns a translated string.

**Standards**

ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**

Any user can use `xa_bqual`.

**See also**

- **Functions**  `xa_gtrid`
- **Stored procedures**  `sp_lock, sp_transactions`
**xa_gtrid**

**Description**
Returns the binary version of the grid component of an ASCII XA transaction ID.

**Syntax**
`xa_gtrid(xactname, int)`

**Parameters**
- `xid` is the ID of an Adaptive Server transaction, obtained from the `xactname` column in `systransactions` or from `sp_transactions`.
- `0` is reserved for future use.

**Examples**

**Example 1**
In this typical situation, returns “0x227f06ca80,” the binary translation of the branch qualifier, and “0xb1946cdc52464a61c4a2fe4e0f5232b,” the global transaction ID, for the Adaptive Server transaction ID “0000000A_IphIT596iC7bF2#AUfkzaM_8DY60E0”:

```sql
1> select xa_gtrid("0000000A_IphIT596iC7bF2#AUfkzaM_8DY60E0", 0)
2> go
```

```
...  
0xb1946cdc52464a61c4a2fe4e0f5232b
(1 row affected)
```

**Example 2**
`xa_bqual` is often used together with `xa_gtrid`. This example returns the global transaction IDs and branch qualifiers from all rows in `systransactions` where its coordinator column is the value of “3”:

```sql
1> select grid=xa_gtrid(xactname, 0),
      bqual=xa_bqual(xactname, 0)
   from systransactions where coordinator = 3
2> go
```

```
grid
bqual

0xb1946cdc52464a61c4a2fe4e0f5232b
```

Reference Manual: Building Blocks 335
If an external transaction is blocked on Adaptive Server and you are using `sp_lock` and `sp_transactions` to identify the blocking transaction, you can use the XA transaction manager to terminate the global transaction. However, when you execute `sp_transactions`, the value of `xactname` it returns is in ASCII string format, while XA Server uses an undecoded binary value. Using `xa_gtrid` thus allows you to determine the `gtrid` portion of the transaction name in a format that can be understood by the XA transaction manager.

`xa_gtrid` returns:

- The translation version of tis string that follows the first “_” (underscore) and preceeds either the second “_” or end-of-string value, whichever comes first.
- NULL if the transaction ID cannot be decoded, or is in an unexpected format.

**Note** `xa_gtrid` does not perform a validation check on the `xid`, but only returns a translated string.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can use `xa_gtrid`.

**See also**
*Functions*  [xa_bqual](#)

*Stored procedures*  `sp_lock`, `sp_transactions`
xmltable

Description
Extracts data from an XML document and returns it as a SQL table.

Syntax
```
xmltable_expression ::= xmltable
( row_pattern passing xml_argument
columns column_definitions
options_parameter)
```
```
row_pattern ::= character_string_literal
xml_argument ::= xml_expression | column_reference |
variable_reference
```
```
column_definitions ::= column_definition [ { , column_definition } ]
column_definition ::= ordinality_column | regular_column
```
```
ordinality_column ::= column_name datatype for ordinality
regular_column ::= column_name datatype [ default literal ] [ null | not null]
 [ path column_pattern ]
column_pattern ::= character_string_literal
options_parameter ::= [ , ] option option_string
option_string ::= basic_string_expression
```

Derived table syntax  Returns a SQL table from within a SQL from clause.
```
from_clause ::= from table_reference [, table_reference]...
table_reference ::= table_view_name | ANSI_join | derived_table
table_view_name::=See the select command in Reference Manual
Volume 2, “Commands.”
ANSI_join::=See the select command in Reference Manual
Volume 2, “Commands.”
derived_table ::= (subquery) as table_name [ (column_name [, column_name]...)]
xmltable_expression as table_name
```

Parameters
xml_argument
is an expression, column reference, or variable, referring to an XML
document.

for
is a reserved XML keyword.

ordinality
is a non-reserved XML keyword.

passing
is a non-reserved XML keyword.
xmltable

**row_pattern**
is an XPath query expression whose result is a sequence of elements from the specified document. The `xmltable` call returns a table with one row for each element in the sequence.

columns
is a non-reserved XML keyword.

column_name
is the user-specified name of the column.

column_pattern
is an XPath query expression that applies to an element of the sequence returned by the `row_pattern`, to extract the data for a column of the result table. If the `column_pattern` is omitted, the `column_pattern` defaults to the `column_name`.

**ordinality_column**
is a column of datatypes integer, smallint, tinyint, decimal, or numeric, which indicates ordering of the elements in the input XML document.

**regular_column**
is any column that is not an ordinality column.

derived_table
is a parenthesized subquery specified in the `from` clause of a SQL query.

path
is a reserved XML keyword.

option
is an `option_string`, defined in `XML Services`, and a reserved XML keyword.

**Examples**

**Example 1** Shows a simple `xmltable` call with the document specified as a character-string literal:

```sql
select * from xmltable('/doc/item'
  passing '<doc><item><id>1</id><name>Box</name></item>'
  + '<item><id>2</id><name>Jar</name></item></doc>'
  columns id int path 'id', name varchar(20) path 'name') as items_table

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Box</td>
</tr>
<tr>
<td>2</td>
<td>Jar</td>
</tr>
</tbody>
</table>

(2 rows affected)
```

**Example 2** Stores the document in a Transact-SQL variable, and references that variable in the `xmltable` call:
declare @doc varchar(16384)
set @doc='<doc><item><id>1</id><name>Box</name></item>'
  +'<item><id>2</id><name>Jar</name></item></doc>

select * from xmltable('/doc/item' passing @doc
  columns id int path 'id', name varchar(20) path 'name') as items_table
id  name
---- -----
1   Box
2   Jar
(2 rows affected)

**Example 3** Stores the document in a table and references it with a subquery:

```sql
select 100 as doc_id,
  '<doc><item><id>1</id><name>Box</name></item><item><id>2</id>
  <name>Jar</name></item></doc>' as doc
into #sample_docs
select * from xmltable('/doc/item'
  passing (select doc from #sample_docs where doc_id=100)
  columns id int path 'id', name varchar(20) path 'name') as items_table
id  name
---- -----
1   Box
2   Jar
(2 rows affected)
```

**Example 4** If a row pattern returns an empty sequence, the result is an empty table:

```sql
select * from xmltable('/doc/item_entry'
  passing '<doc><item><id>1</id><name>Box</name></item>'
  +'<item><id>2</id><name>Jar</name></item></doc>'
  columns id int path 'id',
   name varchar(20) path 'name') as items_table
id  name
---- -----

(0 rows affected)
```

**Example 5** The arguments following the `columns` keyword comprise the list of column definitions. Each column definition specifies a column name and datatype, as in `create table`, and a path, called the column pattern.
When the data for a column is contained in an XML attribute, specify the column pattern using "@" to reference an attribute. For example:

```sql
select * from xmltable ('/doc/item'
  passing '<doc><item id="1"><name>Box</name></item>
    +'<item id="2"><name>Jar</name></item></doc>'
columns id int path '@id', name varchar(20)) as items_table
```

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Box</td>
</tr>
<tr>
<td>2</td>
<td>Jar</td>
</tr>
</tbody>
</table>

(2 rows affected)

**Example 6** A column-pattern is commonly the same as the specified column_name, for example name. In this case, omitting the column-pattern results in defaulting to the column_name:

```sql
select * from xmltable ('/doc/item'
  passing '<doc><item><id>1</id><name>Box</name></item>
    +'<item><id>2</id><name>Jar</name></item></doc>'
columns id int, name varchar(20)) as items_table
```

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Box</td>
</tr>
<tr>
<td>2</td>
<td>Jar</td>
</tr>
</tbody>
</table>

(2 rows affected)

**Example 7** If you want a column pattern to default to the column name, in a column whose value is in an XML attribute, use a quoted identifier. You must then quote such identifiers when you reference them in the results:

```sql
set quoted_identifier on
select '@id', name from xmltable ('/doc/item'
  passing '<doc><item id="1"><name>Box</name></item>
    +'<item id="2"><name>Jar</name></item></doc>'
columns '@id' int, name varchar(20)) as items_table
```

<table>
<thead>
<tr>
<th>@id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Box</td>
</tr>
<tr>
<td>2</td>
<td>Jar</td>
</tr>
</tbody>
</table>

(2 rows affected)
Example 8 You can also use quoted identifiers to specify column names as default column patterns, using column names that are more complex XPath expressions. For example:

```sql
set quoted_identifier on
select "@id", "name/short", "name/full" from xmltable ('/doc/item'
passing '<doc><item id="1"><name><short>Box</short>
<full>Box, packing, moisture resistant, plain</full>
</name></item>'
+ '<item id="2"><name><short>Jar</short>
<full>Jar, lidded, heavy duty</full>
</name></item>'
columns "@id" int, "name/short" varchar(20), "name/full" varchar(50))
as items_table
```

<table>
<thead>
<tr>
<th>@id</th>
<th>name/short</th>
<th>name/full</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Box</td>
<td>Box, packing, moisture resistant, plain</td>
</tr>
<tr>
<td>2</td>
<td>Jar</td>
<td>Jar, lidded, heavy duty</td>
</tr>
</tbody>
</table>

(2 rows affected)

Example 9 The function `text` is implicit in column patterns. This example does not specify `text` in the column pattern for either the `id` or `name` column:

```sql
select * from xmltable ('/doc/item'
passing '<doc><item><id>1><name>Box</name></item>'
+ '<item><id>2><name>Jar</name></item>'
columns id int path 'id', name varchar(20) path 'name') as items_table
```

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Box</td>
</tr>
<tr>
<td>2</td>
<td>Jar</td>
</tr>
</tbody>
</table>

(2 rows affected)

Example 10 Applying an implicit SQL `converst` statement to the data extracted from the column pattern, derives column values in datatype conversions.

```sql
select * from xmltable ('/emps/emp'
passing '<emps>
<emp><id>1><salary>123.45</salary><hired>1/2/2003</hired></emp>'
+ '<emp><id>2><salary>234.56</salary><hired>2/3/2004</hired></emp>'
columns id int path 'id', salary dec(5,2), hired date) as items_table
```

<table>
<thead>
<tr>
<th>id</th>
<th>salary</th>
<th>hired</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-------</td>
<td>-----------</td>
</tr>
<tr>
<td></td>
<td>-------</td>
<td>-----------</td>
</tr>
</tbody>
</table>
### Example 11
You can use an `ordinality_column` in `xmltable` to record the ordering of elements in the input XML document:

```sql
declare @doc varchar(16384)
set @doc = '<doc><item><id>25</id><name>Box</name></item>
        +<item><id>15</id><name>Jar</name></item></doc>'
select * from xmltable('/doc/item' passing @doc
columns item_order int for ordinality,
       id int path 'id',
       name varchar(20) path 'name') as items_table
order by item_order

<table>
<thead>
<tr>
<th>item_order</th>
<th>id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25</td>
<td>Box</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td>Jar</td>
</tr>
</tbody>
</table>
```

Without the `for ordinality` clause and the `item_order` column, there is nothing in the `id` and `name` columns that indicates that the row of `id` 25 precedes the row of `id` 15. The `for ordinality` clause orders the output SQL rows the same as the ordering of the elements in the input XML document.

The datatype of an ordinality column can be any fixed numeric datatype: `int`, `tinyint`, `bigint`, `numeric`, or `decimal`. `numeric` and `decimal` must have a scale of 0. An ordinality column cannot be `real` or `float`.

### Example 12
Omits the `<name>` element from the second `<item>`. The `name` column allows names to be `NULL` by default.

```sql
select * from xmltable ('/doc/item' passing '<doc><item><id>1</id><name>Box</name></item>
      +<item><id>2</id></item></doc>'
columns id int path 'id', name varchar(20) path 'name')
as items_table

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Box</td>
</tr>
<tr>
<td>2</td>
<td>NULL</td>
</tr>
</tbody>
</table>
```

### Example 13
Omits the `<name>` element from the second `<item>`, and specifies `not null` for the `name` column:

```sql
select * from xmltable ('/doc/item' passing '<doc><item><id>1</id><name>Box</name></item>
      +<item><id>2</id></item></doc>'
columns id int path 'id', name varchar(20) path 'name')
as items_table

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Box</td>
</tr>
<tr>
<td>2</td>
<td>NULL</td>
</tr>
</tbody>
</table>
```

CHAPTER 2  Transact-SQL Functions

Example 14 Adds a default clause to the name column, and omits the <name> elements from the second <item>.

```
select * from xmltable ('/doc/item'
  passing '<doc><item><id>1</id><name>Box</name></item>
  + '<item><id>2</id></item></doc>'
columns id int path 'id', name varchar(20) default '***' path 'name'
) as items_table
id   name
----- --------
1    Box
```

Msg 14847, Level 16, State 1:
Line 1:
XMLTABLE column 0, does not allow null values.

Example 15 Shows SQL commands in which you can use an xmltable call in a derived table expression. This example uses xmltable in a simple select statement:

```
select * from xmltable ('/doc/item'
  passing '<doc><item><id>1</id><name>Box</name></item>
  + '<item><id>2</id><name>Jar</name></item></doc>'
columns id int path 'id',
     name varchar(20) path 'name') as items_table
id   name
--   -----  
1    Box
2    ***
(2 rows affected)
```

Example 16 Uses xmltable in a view definition. It stores a document in a table and references that stored document in a create view statement, using xmltable to extract data from the table:

```
select 100 as doc_id,
```
xmltable

'&lt;doc&gt;&lt;item&gt;&lt;id&gt;1&lt;/id&gt;&lt;name&gt;Box&lt;/name&gt;&lt;/item&gt;'  
  +'&lt;item&gt;&lt;id&gt;2&lt;/id&gt;&lt;name&gt;Jar&lt;/name&gt;&lt;/item&gt;&lt;/doc&gt;' as doc  
into sample_docs  
create view items_table as  
  select * from xmltable('/doc/item'  
    passing (select doc from sample_docs where doc_id=100)  
    columns id int path 'id',  
    name varchar(20) path 'name') as xml_extract  
select * from items_table  
id    name  
----------  
1      Box  
2      Jar  
(2 rows affected)

Example 17  Uses xmltable in a cursor:

declare C cursor for  
select * from xmltable ('/doc/item'  
    passing (select doc from sample_docs where id=100)  
    columns id int path 'id',  
    name varchar(20) path 'name') as items_table  
go  
declare @idvar int  
declare @namevar varchar(20)  
open C  
while @@sqlstatus=0  
begin  
fetch C into @idvar, @namevar  
print 'ID "%1!" NAME "%2!"', @idvar, @namevar  
end  
----------  
ID "1" NAME "Box"  
ID "2" NAME "Jar"  
(2 rows affected)

In applications that require multiple actions for each generated row, such as executing update, insert, or delete from other tables, you can process an xmltable result with a cursor loop. Alternatively, store the xmltable result in a temporary table and process that table with a cursor loop.

Example 18  This example uses xmltable in select into:

select * into #extracted_table  
from xmltable ('/doc/item'  
    passing (select doc from sample_docs where doc_id=100
create table #extracted_data (idcol int, namecol varchar(20))
insert into #extracted_data
select * from xmltable('/doc/item'
passing (select doc from sample_docs where doc_id=100)
columns id int path 'id', name varchar(20) path 'name') as items_table
select * from #extracted_data

Example 19 Uses xmltable in an insert command:

Example 20 Uses xmltable in a subquery. xmltable returns a SQL table, so the subquery must perform either an aggregation or a selection to return a single row and column for the subquery result.

Example 21 Joins an xmltable result with other tables, using either multiple table joins in the from clause, or outer joins:
Extracted Table

```
columns id int path 'id', name varchar(20) path 'name') as a) as extracted_table
where prices.id=extracted_table.id
```

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Box</td>
<td>123.45</td>
</tr>
<tr>
<td>2</td>
<td>Jar</td>
<td>234.56</td>
</tr>
</tbody>
</table>

(2 rows affected)

**Example 22**  Uses **xmltable**, with a lateral reference to a column existing in a preceding table in the same from clause as **xmltable**:

create table deptab (col1 int, col2 image)
insert deptab values (1, '<dept>
    <dept-id>1</dept-id>
    <dept-name>Finance</dept-name>
    <employees>
        <emp><name>John</name><id>e11</id></emp>
        <emp><name>Bela</name><id>e12</id></emp>
        <emp><name>James</name><id>e13</id></emp>
    </employees>
</dept>'
)

insert deptab values (2, '<dept>
    <dept-id>2</dept-id>
    <dept-name>Engineering</dept-name>
    <employees>
        <emp><name>Tom</name><id>e21</id></emp>
        <emp><name>Jeff</name><id>e22</id></emp>
        <emp><name>Mary</name><id>e23</id></emp>
    </employees>
</dept>'
)

select id, empname from deptab, xmltable ('/dept/employees/emp' passing deptab.col2 columns empname varchar (8) path 'name', id varchar (8) path 'id') as sample_tab

<table>
<thead>
<tr>
<th>id</th>
<th>empname</th>
</tr>
</thead>
<tbody>
<tr>
<td>e11</td>
<td>John</td>
</tr>
<tr>
<td>e12</td>
<td>Bela</td>
</tr>
<tr>
<td>e13</td>
<td>James</td>
</tr>
<tr>
<td>e21</td>
<td>Tom</td>
</tr>
<tr>
<td>e22</td>
<td>Jeff</td>
</tr>
<tr>
<td>e23</td>
<td>Mary</td>
</tr>
</tbody>
</table>

(6 rows affected)

**Usage**

- **xmltable** is a built-in, table-valued function.
• The syntax of derived tables requires you to specify a table name, even if you do not reference it. Therefore, each `xmltable` expression must also specify a table name.

• The argument following `passing` is the input XML document.

• The result type of an `xmltable` expression is a SQL table, whose column names and their datatypes are specified by `column_definitions`.

• To process documents, you can apply `xmltable` to the XML document in each row of a table of XML documents.

• These keywords are associated with `xmltable`:
  • Reserved – for, option, `xmltable`, path
  • Not reserved – columns, ordinality, passing

• The expressions in the arguments of an `xmltable` call can reference the column names of preceding tables in the `from` clause containing the `xmltable` call. Only tables that precede the `xmltable` call can be referenced. Such a reference, to a column of a preceding table in the same `from` clause, is called a lateral reference. For example:

```sql
select * from T1, xmltable(...passing T1.C1...) as XT2, xmltable(...passing XT2.C2...)as XT3
```

The reference to T1.C1 in the first `xmltable` call is a lateral reference to column C1 of table T1. The reference to XT2.C2 in the second `xmltable` call is a lateral reference to column C2 of the table generated by the first `xmltable` call.

• You cannot use `xmltable` in the `from` clause of an `update` or `delete` statement. For example, the following statement fails:

```sql
update T set T.C=... from xmltable(...) as T where...
```

• Datatypes in regular columns can be of any SQL datatype.

• To handle XML data whose format is not suitable for a SQL `convert` function, extract the data to a string column (`varchar`, `text`, `image`, `java.lang.String`).

• The extracted XML data for the column must be convertible to the column datatype, or an exception is raised.

• If a column pattern returns an empty result, the action taken depends on the `default` and `{null | not null}` clauses.
- The literal following a `default` in a `regular_column` must be assignable to the datatype of the column.

- There can be no more than one `ordinality_column`; the datatype specified for this variable must be integer, smallint, tinyint, decimal, or numeric. decimal and numeric must have a scale of zero.

- An `ordinality_column`, if one exists, is not nullable.

*Note* This default is different from the default value of `create table`.

- The nullable property of other columns is specified by the `(null | not null)` clause. The default is null.

- The current setting of `set quoted_identifier` applies to the clauses of an `xmltable` expression. For example:
  - If `set quoted_identifier` is on, column names can be quoted identifiers, and string literals in `row_pattern`, `column_pattern`, and default literals must be surrounded with single quotation marks.
  - If `set quoted_identifier` is off, column names cannot be quoted identifiers, and string literals in `row_pattern`, `column_pattern`, and default literals can be surrounded with either single or double quotation marks.

- The general format of the `option_string` is described in “option_strings: general format,” in *XML Services, Adaptive Server 15.0*.

### `xmltable` row and column patterns

- `xmltable` row and column patterns are allowed to be only simple paths. Simple paths in XPath consist only of forward traversals using `/` and element/attribute names.

- If the `row_pattern` does not begin at the root level of the document specified by `xml_argument`, an exception is raised. The row pattern must begin at the root of the XML document.

- The row pattern expression cannot contain an XPath function.

- A column pattern must be a relative path.

- If the `row_pattern` specifies an XML function, an exception is raised. The row pattern cannot specify an XML function.

- If a `column_definition` does not specify a path, the default `column_pattern` is the `column_name` of the column definition. This default is subject to the case sensitivity of the server. For example, consider this statement:
select * from xmltable(...columns name varchar(30),...)

If the server is case-insensitive, this is equivalent to the following:

select * from xmltable(...columns name varchar(30)
path 'name',...)

If the server is case-sensitive, the first statement is equivalent to:

select * from xmltable
(...columns name varchar(30)path 'NAME',...)

Generating the rows of the result table

- The result value of an xmltable expression is a T-SQL table RT, defined as follows:
- RT has a row for each element in the XML sequence that results from applying the row_pattern to the xml_argument.
- The rows of RT have a column for each column_definition, with the column_name and datatype specified in the column_definition.
- If a column_definition is a ordinality_column, its value for the Nth row is the integer N.
- If a column_definition is a regular_column, its value for the Nth row corresponds to the following:
  - Let XVAL be the result of applying this XPath expression to the xml_argument:
    
    (row_pattern[N])/column_pattern/text()
  - If XVAL is empty, and the column_definition contains a default clause, the value of the column is that default value.
  - If XVAL is empty and the column_definition specifies not null, an exception is raised.
  - Otherwise, the value of the column is the null value.
- If XVAL is not empty, and the datatype of the column is char, varchar, text, unitext, unichar, univarchar, or java.lang.String, de-entitize XVAL.
- The value of the column is the result of:
  
  convert(datatype,XVAL)
### year

**Description**
Returns an integer that represents the year in the datepart of a specified date.

**Syntax**
```
year(date_expression)
```

**Parameters**
- `date_expression` is an expression of type datetime, smalldatetime, date, time or a character string in a datetime format.

**Examples**
Returns the integer 03:
```
year("11/02/03")
```
```
--------
03
(1 row(s) affected)
```

**Usage**
`year(date_expression)` is equivalent to `datepart(yy, date_expression)`.

**Standards**
ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Any user can execute `year`.

**See also**
- **Datatypes**  
  datetime, smalldatetime, date  
- **Functions**  
  datepart, day, month
CHAPTER 3

Global Variables

Adaptive Server global variables

Global variables are system-defined variables updated by Adaptive Server while the system is running. Some global variables are session-specific, while others are server instance-specific. For example, @@error contains the last error number generated by the system for a given user connection.

See `get_appcontext` and `set_appcontext` to specify application context variables.

To view the value for any global variable, enter:

```
select variable_name
```

For example:

```
select @@char_convert
```

Table 3-1 lists the global variables available for Adaptive Server:

<table>
<thead>
<tr>
<th>Global variable</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>@@active_instances</td>
<td>Returns the number of active instances in the cluster</td>
</tr>
<tr>
<td>@@authmech</td>
<td>A read-only variable that indicates the mechanism used to authenticate the user.</td>
</tr>
<tr>
<td>@@bootcount</td>
<td>Returns the number of times an Adaptive Server installation has been booted.</td>
</tr>
<tr>
<td>@@boottime</td>
<td>Returns the date and time Adaptive Server was last booted.</td>
</tr>
<tr>
<td>@@bulkarraysize</td>
<td>Returns the number of rows to be buffered in local server memory before being transferred using the bulk copy interface. Used only with Component Integration Services for transferring rows to a remote server using <code>select into</code>. For more information, see the Component Integration Services User's Guide.</td>
</tr>
</tbody>
</table>

Table 3-1: Adaptive Server global variables
## Adaptive Server global variables

<table>
<thead>
<tr>
<th>Global variable</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>@@bulkbatchsize</code></td>
<td>Returns the number of rows transferred to a remote server via <code>select into proxy_table</code> using the bulk interface. Used only with Component Integration Services for transferring rows to a remote server using <code>select into</code>. For more information, see the <a href="#">Component Integration Services User’s Guide</a>.</td>
</tr>
<tr>
<td><code>@@char_convert</code></td>
<td>Returns 0 if character set conversion is not in effect. Returns 1 if character set conversion is in effect.</td>
</tr>
<tr>
<td><code>@@cis_rpc_handling</code></td>
<td>Returns 0 if cis rpc handling is off. Returns 1 if cis rpc handling is on. For more information, see the <a href="#">Component Integration Services User’s Guide</a>.</td>
</tr>
<tr>
<td><code>@@cis_version</code></td>
<td>Returns the expansion factor used when converting from the server character set to the client character set. For example, if it contains a value of 2, a character in the server character set could take up to twice the number of bytes after translation to the client character set.</td>
</tr>
<tr>
<td><code>@@client_csexpansion</code></td>
<td>Returns -1 if the client character set has never been initialized; returns the client character set ID from <code>syscharsets</code> for the connection if the client character set has been initialized.</td>
</tr>
<tr>
<td><code>@@client_csname</code></td>
<td>Returns NULL if client character set has never been initialized; returns the name of the character set for the connection if the client character set has been initialized.</td>
</tr>
<tr>
<td><code>@@clusterboottime</code></td>
<td>Returns the date and time the cluster was first started, even if the instance that originally started the cluster start has shut down.</td>
</tr>
<tr>
<td><code>@@clustercoordid</code></td>
<td>Returns the instance id of the current cluster coordinator.</td>
</tr>
<tr>
<td><code>@@clustermode</code></td>
<td>Returns the string: “shared-disk cluster”</td>
</tr>
<tr>
<td><code>@@clustername</code></td>
<td>Returns the name of the cluster.</td>
</tr>
<tr>
<td><code>@@cmpstate</code></td>
<td>Returns the current mode of Adaptive Server in a high availability environment. Not used in a non-high-availability environment.</td>
</tr>
<tr>
<td><code>@@connections</code></td>
<td>Returns the number of user logins attempted.</td>
</tr>
<tr>
<td><code>@@cpu_busy</code></td>
<td>Returns the amount of time, in ticks, that the CPU has spent doing Adaptive Server work since the last time Adaptive Server was started.</td>
</tr>
</tbody>
</table>
## CHAPTER 3  Global Variables

<table>
<thead>
<tr>
<th>Global variable</th>
<th>Definition</th>
</tr>
</thead>
</table>
| @@cursor_rows   | A global variable designed specifically for scrollable cursors. Displays the total number of rows in the cursor result set. Returns the following values:  
  • -1 – the cursor is:  
    • Dynamic – because dynamic cursors reflect all changes, the number of rows that qualify for the cursor is constantly changing. You can never be certain that all the qualified rows are retrieved.  
    • semi_sensitive and scrollable, but the scrolling worktable is not yet fully populated – the number of rows that qualify the cursor is unknown at the time this value is retrieved.  
  • 0 – either no cursors are open, no rows qualify for the last opened cursor, or the last open cursor is closed or deallocated.  
  • n – the last opened or fetched cursor result set is fully populated. The value returned is the total number of rows in the cursor result set. |
| @@curloid       | Either no cursors are open, no rows qualify for the last opened cursor, or the last open cursor is closed or deallocated. |
| @@datefirst     | Set using `set datefirst n` where n is a value between 1 and 7. Returns the current value of @@datefirst, indicating the specified first day of each week, expressed as tinyint. The default value in Adaptive Server is Sunday (based on the us_language default), which you set by specifying `set datefirst 7`. See the datefirst option of the `set` command for more information on settings and values. |
| @@dbts          | Returns the timestamp of the current database. |
| @@error         | Returns the error number most recently generated by the system. |
| @@errorlog      | Returns the full path to the directory in which the Adaptive Server error log is kept, relative to $SYBASE directory (%SYBASE% on NT). |
| @@failedoverconn| Returns a value greater than 0 if the connection to the primary companion has failed over and is executing on the secondary companion server. Used only in a high availability environment, and is session-specific. |
| @@fetch_status  | Returns:  
  • 0 – fetch operation successful  
  • -1 – fetch operation unsuccessful  
  • -2 – value reserved for future use |
| @@guestuserid   | Returns the ID of the guest user. |
| @@hacmpservername | Returns the name of the companion server in a high availability setup. |
| @@haconnection  | Returns a value greater than 0 if the connection has the failover property enabled. This is a session-specific property. |
| @@heapmemsize   | Returns the size of the heap memory pool, in bytes. See the System Administration Guide for more information on heap memory. |
| @@identity      | Returns the most recently generated IDENTITY column value. |
## Adaptive Server global variables

<table>
<thead>
<tr>
<th>Global variable</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>@@idle</td>
<td>Returns the amount of time, in ticks, that Adaptive Server has been idle since it was last started.</td>
</tr>
<tr>
<td>@@instanceid</td>
<td>Returns the id of the instance from which it was executed.</td>
</tr>
<tr>
<td>@@instancetype</td>
<td>Returns the name of the instance from which it was executed.</td>
</tr>
<tr>
<td>@@invaliduserid</td>
<td>Returns a value of -1 for an invalid user ID.</td>
</tr>
<tr>
<td>@@io_busy</td>
<td>Returns the amount of time, in ticks, that Adaptive Server has spent doing input and output operations.</td>
</tr>
<tr>
<td>@@isolation</td>
<td>Returns the value of the session-specific isolation level (0, 1, or 3) of the current Transact-SQL program.</td>
</tr>
<tr>
<td>@@jsinstanceid</td>
<td>ID of the instance on which the Job Scheduler is running, or will run once enabled.</td>
</tr>
<tr>
<td>@@kernel_addr</td>
<td>Returns the starting address of the first shared memory region that contains the kernel region. The result is in the form of 0xaddress pointer value.</td>
</tr>
<tr>
<td>@@kernel_size</td>
<td>Returns the size of the kernel region that is part of the first shared memory region.</td>
</tr>
<tr>
<td>@@langid</td>
<td>Returns the server-wide language ID of the language in use, as specified in syslanguages.langid.</td>
</tr>
<tr>
<td>@@language</td>
<td>Returns the name of the language in use, as specified in syslanguages.name.</td>
</tr>
<tr>
<td>@@lastlogindate</td>
<td>Available to each user login session, @@lastlogindate includes a datetime datatype, its value is the lastlogindate column for the login account before the current session was established. This variable is specific to each login session and can be used by that session to determine the previous login to the account. If the account has not been used previously or “sp_passwordpolicy ‘set’, enable last login updates” is 0, then the value of @@lastlogindate is NULL.</td>
</tr>
<tr>
<td>@@lock_timeout</td>
<td>Set using set lock wait n. Returns the current lock_timeout setting, in milliseconds. @@lock_timeout returns the value of n. The default value is no timeout. If no set lock wait n is executed at the beginning of the session, @@lock_timeout returns -1.</td>
</tr>
<tr>
<td>@@maxcharlen</td>
<td>Returns the maximum length, in bytes, of a character in Adaptive Server’s default character set.</td>
</tr>
<tr>
<td>@@max_connections</td>
<td>Returns the maximum number of simultaneous connections that can be made with Adaptive Server in the current computer environment. You can configure Adaptive Server for any number of connections less than or equal to the value of @@max_connections with the number of user connections configuration parameter.</td>
</tr>
<tr>
<td>@@maxgroupid</td>
<td>Returns the highest group user ID. The highest value is 1048576.</td>
</tr>
<tr>
<td>@@maxpagesize</td>
<td>Returns the server’s logical page size.</td>
</tr>
<tr>
<td>@@max_precision</td>
<td>Returns the precision level used by decimal and numeric datatypes set by the server. This value is a fixed constant of 38.</td>
</tr>
<tr>
<td>@@maxspid</td>
<td>Returns maximum valid value for the spid.</td>
</tr>
<tr>
<td>@@maxsuid</td>
<td>Returns the highest server user ID. The default value is 2147483647.</td>
</tr>
<tr>
<td>@@maxuserid</td>
<td>Returns the highest user ID. The highest value is 2147483647.</td>
</tr>
<tr>
<td>@@mempool_addr</td>
<td>Returns the global memory pool table address. The result is in the form 0xaddress pointer value. This variable is for internal use.</td>
</tr>
</tbody>
</table>
### Global Variable Definition

<table>
<thead>
<tr>
<th><strong>Global variable</strong></th>
<th><strong>Definition</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>@@min_poolsize</code></td>
<td>Returns the minimum size of a named cache pool, in kilobytes. It is calculated based on the DEFAULT_POOL_SIZE, which is 256, and the current value of max database page size.</td>
</tr>
<tr>
<td><code>@@mingroupid</code></td>
<td>Returns the lowest group user ID. The lowest value is 16384.</td>
</tr>
<tr>
<td><code>@@minspid</code></td>
<td>Returns 1, which is the lowest value for spid.</td>
</tr>
<tr>
<td><code>@@minssid</code></td>
<td>Returns the minimum server user ID. The lowest value is -32768.</td>
</tr>
<tr>
<td><code>@@minuserid</code></td>
<td>Returns the lowest user ID. The lowest value is -32768.</td>
</tr>
<tr>
<td><code>@@monitors_active</code></td>
<td>Reduces the number of messages displayed by sp_sysmon.</td>
</tr>
<tr>
<td><code>@@ncharsize</code></td>
<td>Returns the maximum length, in bytes, of a character set in the current server default character set.</td>
</tr>
<tr>
<td><code>@@nestlevel</code></td>
<td>Returns the current nesting level.</td>
</tr>
<tr>
<td><code>@@nodeid</code></td>
<td>Returns the current installation's 48-bit node identifier. Adaptive Server generates a nodeid the first time the master device is first used, and uniquely identifies an Adaptive Server installation.</td>
</tr>
<tr>
<td><code>@@optgoal</code></td>
<td>Returns the current optimization goal setting for query optimization</td>
</tr>
<tr>
<td><code>@@options</code></td>
<td>Returns a hexadecimal representation of the session’s set options.</td>
</tr>
<tr>
<td><code>@@opttimeoutlimit</code></td>
<td>Returns the current optimization timeout limit setting for query optimization</td>
</tr>
<tr>
<td><code>@@pack_received</code></td>
<td>Retruns the number of input packets read by Adaptive Server.</td>
</tr>
<tr>
<td><code>@@pack_sent</code></td>
<td>Returns the number of output packets written by Adaptive Server.</td>
</tr>
<tr>
<td><code>@@packet_errors</code></td>
<td>Returns the number of errors detected by Adaptive Server while reading and writing packets.</td>
</tr>
<tr>
<td><code>@@pagesize</code></td>
<td>Returns the server’s virtual page size.</td>
</tr>
<tr>
<td><code>@@parallel_degree</code></td>
<td>Returns the current maximum parallel degree setting.</td>
</tr>
<tr>
<td><code>@@probesuid</code></td>
<td>Returns a value of 2 for the probe user ID.</td>
</tr>
<tr>
<td><code>@@procid</code></td>
<td>Returns the stored procedure ID of the currently executing procedure.</td>
</tr>
<tr>
<td><code>@@quorum_physname</code></td>
<td>Returns the physical path for the quorum device</td>
</tr>
<tr>
<td><code>@@recovery_state</code></td>
<td>Indicates whether Adaptive Server is in recovery based on these returns:</td>
</tr>
<tr>
<td></td>
<td>• NOT_IN_RECOVERY – Adaptive Server is not in startup recovery or in failover recovery. Recovery has been completed and all databases that can be online are brought online.</td>
</tr>
<tr>
<td></td>
<td>• RECOVERY_TUNING – Adaptive Server is in recovery (either startup or failover) and is tuning the optimal number of recovery tasks.</td>
</tr>
<tr>
<td></td>
<td>• BOOTTIME_RECOVERY – Adaptive Server is in startup recovery and has completed tuning the optimal number of tasks. Not all databases have been recovered.</td>
</tr>
<tr>
<td></td>
<td>• FAILOVER_RECOVER – Adaptive Server is in recovery during an HA failover and has completed tuning the optimal number of recovery tasks. All databases are not brought online yet.</td>
</tr>
<tr>
<td><code>@@repartition_degree</code></td>
<td>Returns the current dynamic repartitioning degree setting</td>
</tr>
</tbody>
</table>
Adaptive Server global variables

<table>
<thead>
<tr>
<th>Global variable</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>@@resource_granularity</td>
<td>Returns the maximum resource usage hint setting for query optimization.</td>
</tr>
<tr>
<td>@@rowcount</td>
<td>Returns the number of rows affected by the last query. The value of @@rowcount is affected by whether the specified cursor is forward-only or scrollable. If the cursor is the default, non-scrollable cursor, the value of @@rowcount increments one by one, in the forward direction only, until the number of rows in the result set are fetched. These rows are fetched from the underlying tables to the client. The maximum value for @@rowcount is the number of rows in the result set. In the default cursor, @@rowcount is set to 0 by any command that does not return or affect rows, such as an if or set command, or an update or delete statement that does not affect any rows. If the cursor is scrollable, there is no maximum value for @@rowcount. The value continues to increment with each fetch, regardless of direction, and there is no maximum value. The @@rowcount value in scrollable cursors reflects the number of rows fetched from the result set, not from the underlying tables, to the client.</td>
</tr>
<tr>
<td>@@scan_parallel_degree</td>
<td>Returns the current maximum parallel degree setting for nonclustered index scans.</td>
</tr>
<tr>
<td>@@servername</td>
<td>Returns the name of Adaptive Server.</td>
</tr>
<tr>
<td>@@setrowcount</td>
<td>Returns the current value for set rowcount.</td>
</tr>
<tr>
<td>@@shmem_flags</td>
<td>Returns the shared memory region properties. This variable is for internal use. There are a total of 13 different properties values corresponding to 13 bits in the integer. The valid values represented from low to high bit are: MR_SHARED, MR_SPECIAL, MR_PRIVATE, MR_READABLE, MR_WRITABLE, MR_EXECUTABLE, MR_HWCOHERENCY, MR_SWCOHERENCY, MR_EXACT, MR_BEST, MR_NAIL, MR_PSUEDO, MR_ZERO.</td>
</tr>
<tr>
<td>@@spid</td>
<td>Returns the server process ID of the current process.</td>
</tr>
<tr>
<td>@@sqlstatus</td>
<td>Returns status information (warning exceptions) resulting from the execution of a fetch statement.</td>
</tr>
<tr>
<td>@@ssl_ciphersuite</td>
<td>Returns NULL if SSL is not used on the current connection; otherwise, it returns the name of the cipher suite you chose during the SSL handshake on the current connection.</td>
</tr>
<tr>
<td>@@stringsize</td>
<td>Returns the amount of character data returned from a toString() method. The default is 50. Max values may be up to 2GB. A value of zero specifies the default value. See the Component Integration Services User's Guide for more information.</td>
</tr>
<tr>
<td>@@system_busy</td>
<td>Number of ticks during which Adaptive Server was running a system task.</td>
</tr>
<tr>
<td>@@sys_tempdbid</td>
<td>Returns the database id of the executing instance’s effective local system temporary database.</td>
</tr>
<tr>
<td>@@system_view</td>
<td>Returns the session-specific system view setting, either “instance” or “cluster”.</td>
</tr>
<tr>
<td>@@tempdbid</td>
<td>Returns a valid temporary database ID (dbid) of the session’s assigned temporary database.</td>
</tr>
<tr>
<td>@@textcolid</td>
<td>Returns the column ID of the column referenced by @@textptr.</td>
</tr>
</tbody>
</table>
### Global Variables

<table>
<thead>
<tr>
<th>Global variable</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>@@textdataptnid</code></td>
<td>Returns the partition ID of a text partition containing the column referenced by <code>@@textptr</code>.</td>
</tr>
<tr>
<td><code>@@textdbid</code></td>
<td>Returns the database ID of a database containing an object with the column referenced by <code>@@textptr</code>.</td>
</tr>
<tr>
<td><code>@@textobjid</code></td>
<td>Returns the object ID of an object containing the column referenced by <code>@@textptr</code>.</td>
</tr>
<tr>
<td><code>@@textpnmid</code></td>
<td>Returns the partition ID of a data partition containing the column referenced by <code>@@textptr</code>.</td>
</tr>
<tr>
<td><code>@@textptr</code></td>
<td>Returns the text pointer of the last text, unitext, or image column inserted or updated by a process (Not the same as the <code>textptr</code> function).</td>
</tr>
<tr>
<td><code>@@textptr_parameters</code></td>
<td>Returns 0 if the current status of the <code>textptr_parameters</code> configuration parameter is off. Returns 1 if the current status of the <code>textptr_parameters</code> if on. See the Component Integration Services User's Guide for more information.</td>
</tr>
<tr>
<td><code>@@textsize</code></td>
<td>Returns the limit on the number of bytes of text, unitext, or image data a select returns. Default limit is 32K bytes for isql; the default depends on the client software. Can be changed for a session with <code>set textsize</code>.</td>
</tr>
<tr>
<td><code>@@textts</code></td>
<td>Returns the text timestamp of the column referenced by <code>@@textptr</code>.</td>
</tr>
<tr>
<td><code>@@thresh_hysteresis</code></td>
<td>Returns the decrease in free space required to activate a threshold. This amount, also known as the hysteresis value, is measured in 2K database pages. It determines how closely thresholds can be placed on a database segment.</td>
</tr>
<tr>
<td><code>@@timeticks</code></td>
<td>Returns the number of microseconds per tick. The amount of time per tick is machine-dependent.</td>
</tr>
<tr>
<td><code>@@total_errors</code></td>
<td>Returns the number of errors detected by Adaptive Server while reading and writing.</td>
</tr>
<tr>
<td><code>@@total_read</code></td>
<td>Returns the number of disk reads by Adaptive Server.</td>
</tr>
<tr>
<td><code>@@total_write</code></td>
<td>Returns the number of disk writes by Adaptive Server.</td>
</tr>
<tr>
<td><code>@@tranchained</code></td>
<td>Returns 0 if the current transaction mode of the Transact-SQL program is unchained. Returns 1 if the current transaction mode of the Transact-SQL program is chained.</td>
</tr>
<tr>
<td><code>@@trancount</code></td>
<td>Returns the nesting level of transactions in the current user session.</td>
</tr>
<tr>
<td><code>@@transactional_rpc</code></td>
<td>Returns 0 if RPCs to remote servers are transactional. Returns 1 if RPCs to remote servers are not transactional. For more information, see enable xact coordination and set option transactional_rpc in the Reference Manual. Also, see the Component Integration Services User's Guide.</td>
</tr>
<tr>
<td><code>@@transtate</code></td>
<td>Returns the current state of a transaction after a statement executes in the current user session.</td>
</tr>
<tr>
<td><code>@@unicharsize</code></td>
<td>Returns 2, the size of a character in unichar.</td>
</tr>
<tr>
<td><code>@@user_busy</code></td>
<td>Number of ticks during which Adaptive Server was running a user task.</td>
</tr>
<tr>
<td><code>@@version</code></td>
<td>Returns the date, version string, and so on of the current release of Adaptive Server.</td>
</tr>
<tr>
<td><code>@@version_number</code></td>
<td>Returns the whole version of the current release of Adaptive Server as an integer.</td>
</tr>
<tr>
<td><code>@@version_as_integer</code></td>
<td>Returns the number of the last upgrade version of the current release of Adaptive Server as an integer. For example, <code>@@version_as_integer</code> returns 12500 if you are running Adaptive Server version 12.5, 12.5.0.3, or 12.5.1.</td>
</tr>
</tbody>
</table>
Adaptive Server global variables

The value of @@user_busy + @@system_busy should equal the value of @@cpu_busy

Using global variables in a clustered environment

For @@servername, the Cluster Edition returns the name of the cluster, not the instance name. Use @@instancename to return the name of the instance.

In a non-clustered Adaptive Server environment, the value for @@identity changes for every record inserted. If the most recent record inserted contains a column with the IDENTITY property, @@identity is set to the value of this column, otherwise it is set to "0" (an invalid value). This variable is session-specific, and takes its value based on the last insert that occurred during this session.

In a clustered environment, multiple nodes perform inserts on tables, so the session-specific behavior is not retained for @@identity. In a clustered environment, the value for @@identity depends on the last record inserted in the node for the current session and not on the last record inserted in the cluster.
This chapter describes Transact-SQL expressions, valid identifiers, and wildcard characters.

Topics covered are:

<table>
<thead>
<tr>
<th>Topics</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expressions</td>
<td>359</td>
</tr>
<tr>
<td>Identifiers</td>
<td>369</td>
</tr>
<tr>
<td>Pattern matching with wildcard characters</td>
<td>377</td>
</tr>
</tbody>
</table>

**Expressions**

An expression is a combination of one or more constants, literals, functions, column identifiers and/or variables, separated by operators, that returns a single value. Expressions can be of several types, including arithmetic, relational, logical (or Boolean), and character string. In some Transact-SQL clauses, a subquery can be used in an expression. A case expression can be used in an expression.

Table 4-1 lists the types of expressions that are used in Adaptive Server syntax statements.

**Table 4-1: Types of expressions used in syntax statements**

<table>
<thead>
<tr>
<th>Usage</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression</td>
<td>Can include constants, literals, functions, column identifiers, variables, or parameters</td>
</tr>
<tr>
<td>logical expression</td>
<td>An expression that returns TRUE, FALSE, or UNKNOWN</td>
</tr>
<tr>
<td>constant expression</td>
<td>An expression that always returns the same value, such as “5+3” or “ABCDE”</td>
</tr>
<tr>
<td>float_expr</td>
<td>Any floating-point expression or an expression that implicitly converts to a floating value</td>
</tr>
<tr>
<td>integer_expr</td>
<td>Any integer expression or an expression that implicitly converts to an integer value</td>
</tr>
<tr>
<td>numeric_expr</td>
<td>Any numeric expression that returns a single value</td>
</tr>
<tr>
<td>char_expr</td>
<td>Any expression that returns a single character-type value</td>
</tr>
<tr>
<td>binary_expression</td>
<td>An expression that returns a single binary or varbinary value</td>
</tr>
</tbody>
</table>
Expressions

Size of expressions

Expressions returning binary or character datum can be up to 16384 bytes in length. However, earlier versions of Adaptive Server only allowed expressions to be up to 255 bytes in length. If you have upgraded from an earlier release of Adaptive Server, and your stored procedures or scripts store a result string of up to 255 bytes, the remainder will be truncated. You may have to re-write these stored procedures and scripts for to account for the additional length of the expressions.

Arithmetic and character expressions

The general pattern for arithmetic and character expressions is:

```
{ constant | column_name | function | (subquery)
| (case_expression)}
[[arithmetic_operator | bitwise_operator |
  string_operator | comparison_operator ]
{ constant | column_name | function | (subquery)
| case_expression}]
...
```

Relational and logical expressions

A logical expression or relational expression returns TRUE, FALSE, or UNKNOWN. The general patterns are:

```
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
```
CHAPTER 4  Expressions, Identifiers, and Wildcard Characters

Operator precedence

Operators have the following precedence levels, where 1 is the highest level and 6 is the lowest:

1  unary (single argument) – + ~
2  * / %
3  binary (two argument) + – & | ^
4  not
5  and
6  or

When all operators in an expression are at the same level, the order of execution is left to right. You can change the order of execution with parentheses—the most deeply nested expression is processed first.

Arithmetic operators

Adaptive Server uses the following arithmetic operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>–</td>
<td>Subtraction</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
</tr>
<tr>
<td>%</td>
<td>Modulo (Transact-SQL extension)</td>
</tr>
</tbody>
</table>

Addition, subtraction, division, and multiplication can be used on exact numeric, approximate numeric, and money type columns.

The modulo operator cannot be used on smallmoney, money, numeric, float or real columns. Modulo finds the integer remainder after a division involving two whole numbers. For example, 21 % 11 = 10 because 21 divided by 11 equals 1 with a remainder of 10.

When you perform arithmetic operations on mixed datatypes, for example float and int, Adaptive Server follows specific rules for determining the type of the result. For more information, see Chapter 1, “System and User-Defined Datatypes.”
Expressions

Bitwise operators

The bitwise operators are a Transact-SQL extension for use with integer type data. These operators convert each integer operand into its binary representation, then evaluate the operands column by column. A value of 1 corresponds to true; a value of 0 corresponds to false.

Table 4-3 summarizes the results for operands of 0 and 1. If either operand is NULL, the bitwise operator returns NULL:

<table>
<thead>
<tr>
<th>Table 4-3: Truth tables for bitwise operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp; (and)</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>^ (exclusive or)</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>~ (not)</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

The examples in Table 4-4 use two tinyint arguments, A = 170 (10101010 in binary form) and B = 75 (01001011 in binary form).
Table 4-4: Examples of bitwise operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Binary form</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A &amp; B)</td>
<td>10101010</td>
<td>10</td>
<td>Result column equals 1 if both A and B are 1. Otherwise, result column equals 0.</td>
</tr>
<tr>
<td></td>
<td>01001011</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>------------</td>
<td>00001010</td>
<td></td>
</tr>
<tr>
<td>(A</td>
<td>B)</td>
<td>10101010</td>
<td>235</td>
</tr>
<tr>
<td></td>
<td>01001011</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>------------</td>
<td>11000011</td>
<td></td>
</tr>
<tr>
<td>(A ^ B)</td>
<td>10101010</td>
<td>225</td>
<td>Result column equals 1 if either A or B, but not both, is 1</td>
</tr>
<tr>
<td></td>
<td>01001011</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>------------</td>
<td>11000001</td>
<td></td>
</tr>
<tr>
<td>(~A)</td>
<td>10101010</td>
<td>85</td>
<td>All 1s are changed to 0s and all 0s to 1s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>01010101</td>
<td></td>
</tr>
</tbody>
</table>

String concatenation operator

You can use both the + and || (double-pipe) string operators to concatenate two or more character or binary expressions. For example, the following displays author names under the column heading Name in last-name first-name order, with a comma after the last name; for example, “Bennett, Abraham.”:

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

This example results in "abcdef", "abcdef":

```sql
select "abc" + "def", "abc" || "def"
```

The following returns the string “abc def”. The empty string is interpreted as a single space in all char, varchar, unichar, nchar, nvarchar, and text concatenation, and in varchar and univarchar insert and assignment statements:

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

When concatenating non-character, non-binary expressions, always use convert:

```sql
select "The date is " + convert(varchar(12), getdate())
```
A string concatenated with NULL evaluates to the value of the string. This is an exception to the SQL standard, which states that a string concatenated with a NULL should evaluate to NULL.

Comparison operators

Adaptive Server uses the comparison operators listed in Table 4-5:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Not equal to</td>
</tr>
<tr>
<td>!=</td>
<td>Transact-SQL extension</td>
</tr>
<tr>
<td>!&gt;</td>
<td>Transact-SQL extension</td>
</tr>
<tr>
<td>!&lt;</td>
<td>Transact-SQL extension</td>
</tr>
</tbody>
</table>

In comparing character data, < means closer to the beginning of the server’s sort order and > means closer to the end of the sort order. Uppercase and lowercase letters are equal in a case-insensitive sort order. Use sp_helplist to see the sort order for your Adaptive Server. Trailing blanks are ignored for comparison purposes. So, for example, “Dirk” is the same as “Dirk ”.

In comparing dates, < means earlier and > means later.

Put single or double quotes around all character and datetime data used with a comparison operator:

```sql
= "Bennet"
> "May 22 1947"
```

Nonstandard operators

The following operators are Transact-SQL extensions:

- Modulo operator: %
- Negative comparison operators: !>, !<, !=
• Bitwise operators: ~, ^, |, &
• Join operators: *= and =*

Using *any, all* and *in*

any is used with <, >, or = and a subquery. It returns results when any value retrieved in the subquery matches the value in the where or having clause of the outer statement. For more information, see the *Transact-SQL User's Guide*.

all is used with < or > and a subquery. It returns results when all values retrieved in the subquery are less than (<) or greater than (>) the value in the where or having clause of the outer statement. For more information, see the *Transact-SQL User's Guide*.

in returns results when any value returned by the second expression matches the value in the first expression. The second expression must be a subquery or a list of values enclosed in parentheses. in is equivalent to = any. For more information, see *where clause* in *Reference Manual: Commands*.

Negating and testing

not negates the meaning of a keyword or logical expression.

Use exists, followed by a subquery, to test for the existence of a particular result.

Ranges

between is the range-start keyword; and is the range-end keyword. The following range is inclusive:

\[ \text{where column1 between } x \text{ and } y \]

The following range is not inclusive:

\[ \text{where column1 > } x \text{ and column1 < } y \]

Using nulls in expressions

Use is null or is not null in queries on columns defined to allow null values.
An expression with a bitwise or arithmetic operator evaluates to NULL if any of the operands are null. For example, the following evaluates to NULL if `column1` is NULL:

```
1 + column1
```

**Comparisons that return TRUE**

In general, the result of comparing null values is UNKNOWN, since it is not possible to determine whether NULL is equal (or not equal) to a given value or to another NULL. However, the following cases return TRUE when `expression` is any column, variable or literal, or combination of these, which evaluates as NULL:

- `expression` is null
- `expression = null`
- `expression = @x`, where `@x` is a variable or parameter containing NULL. This exception facilitates writing stored procedures with null default parameters.
- `expression != n`, where `n` is a literal that does not contain NULL, and `expression` evaluates to NULL.

The negative versions of these expressions return TRUE when the expression does not evaluate to NULL:

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

**Note** The far right side of these exceptions is a literal null, or a variable or parameter containing NULL. If the far right side of the comparison is an expression (such as `@nullvar + 1`), the entire expression evaluates to NULL.

Following these rules, null column values do not join with other null column values. Comparing null column values to other null column values in a `where` clause always returns UNKNOWN for null values, regardless of the comparison operator, and the rows are not included in the results. For example, this query returns no result rows where `column1` contains NULL in both tables (although it may return other rows):

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

**Difference between FALSE and UNKNOWN**

Although neither FALSE nor UNKNOWN returns values, there is an important logical difference between FALSE and UNKNOWN, because the opposite of false ("not false") is true. For example, “1 = 2” evaluates to false and its opposite, “1 != 2”, evaluates to true. But “not unknown” is still unknown. If null values are included in a comparison, you cannot negate the expression to get the opposite set of rows or the opposite truth value.

**Using “NULL” as a character string**

Only columns for which NULL was specified in the `create table` statement and into which you have explicitly entered NULL (no quotes), or into which no data has been entered, contain null values. Avoid entering the character string “NULL” (with quotes) as data for a character column. It can only lead to confusion. Use “N/A”, “none”, or a similar value instead. When you want to enter the value NULL explicitly, do not use single or double quotes.

**NULL compared to the empty string**

The empty string (" ") is always stored as a single space in variables and column data. This concatenation statement is equivalent to “abc def”, not to “abcdef”:

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

The empty string is never evaluated as NULL.

**Connecting expressions**

and connects two expressions and returns results when both are true. or connects two or more conditions and returns results when either of the conditions is true.

When more than one logical operator is used in a statement, and is evaluated before or. You can change the order of execution with parentheses.

Table 4-6 shows the results of logical operations, including those that involve null values.
Expressions

Table 4-6: Truth tables for logical expressions

<table>
<thead>
<tr>
<th></th>
<th>TRUE</th>
<th>FALSE</th>
<th>NULL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>UNKNOWN</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>NULL</td>
<td>UNKNOWN</td>
<td>FALSE</td>
<td>UNKNOWN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>TRUE</th>
<th>FALSE</th>
<th>NULL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>UNKNOWN</td>
</tr>
<tr>
<td>NULL</td>
<td>TRUE</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>TRUE</th>
<th>FALSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>FALSE</td>
<td>TRUE</td>
<td></td>
</tr>
<tr>
<td>NULL</td>
<td>UNKNOWN</td>
<td></td>
</tr>
</tbody>
</table>

The result UNKNOWN indicates that one or more of the expressions evaluates to NULL, and that the result of the operation cannot be determined to be either TRUE or FALSE. See “Using nulls in expressions” on page 365 for more information.

Using parentheses in expressions

Parentheses can be used to group the elements in an expression. When “expression” is given as a variable in a syntax statement, a simple expression is assumed. “Logical expression” is specified when only a logical expression is acceptable.

Comparing character expressions

Character constant expressions are treated as varchar. If they are compared with non-varchar variables or column data, the datatype precedence rules are used in the comparison (that is, the datatype with lower precedence is converted to the datatype with higher precedence). If implicit datatype conversion is not supported, you must use the convert function.

Comparison of a char expression to a varchar expression follows the datatype precedence rule; the “lower” datatype is converted to the “higher” datatype. All varchar expressions are converted to char (that is, trailing blanks are appended) for the comparison. If a unichar expression is compared to a char (varchar, nchar, nvarchar) expression, the latter is implicitly converted to unichar.
Using the empty string

The empty string (""") or (""") is interpreted as a single blank in insert or assignment statements on varchar or univarchar data. In concatenation of varchar, char, rechar, nvarchar data, the empty string is interpreted as a single space; for following example is stored as “abc def”:

"abc" + "" + "def"

The empty string is never evaluated as NULL.

Including quotation marks in character expressions

There are two ways to specify literal quotes within a char, or varchar entry. The first method is to double the quotes. For example, if you begin a character entry with a single quote and you want to include a single quote as part of the entry, use two single quotes:

'I don't understand.'

With double quotes:

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

The second method is to enclose a quote in the opposite kind of quote mark. In other words, surround an entry containing a double quote with single quotes (or vice versa). Here are some examples:

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

Using the continuation character

To continue a character string to the next line on your screen, enter a backslash (\) before going to the next line.

Identifiers

Identifiers are names for database objects such as databases, tables, views, columns, indexes, triggers, procedures, defaults, rules, and cursors.
The limit for the length of object names or identifiers is 255 bytes for regular identifiers, and 253 bytes for delimited identifiers. The limit applies to most user-defined identifiers including table name, column name, index name and so on. Due to the expanded limits, some system tables (catalogs) and built-in functions have been expanded.

For variables, “@” count as 1 byte, and the allowed name for it is 254 bytes long.

Listed below are the identifiers, system tables, and built-in functions that are affected these limits.

The maximum length for these identifiers is now 255 bytes.

- Table name
- Column name
- Index name
- View name
- User-defined datatype
- Trigger name
- Default name
- Rule name
- Constraint name
- Procedure name
- Variable name
- JAR name
- Name of LWP or dynamic statement
- Function name
- Name of the time range
- Application context name

Most user-defined Adaptive Server identifiers can be a maximum of 255 bytes in length, whether single-byte or multibyte characters are used. Others can be a maximum of 30 bytes. Refer to the Transact-SQL User’s Guide for a list of both 255-byte and 30-byte identifiers.
The first character of an identifier must be either an alphabetic character, as defined in the current character set, or the underscore (_ ) character.

**Note** Temporary table names, which begin with the pound sign (#), and variable names, which begin with the at sign (@), are exceptions to this rule.

Subsequent characters can include letters, numbers, the symbols #, @, _, and currency symbols such as $ (dollars), ¥ (yen), and £ (pound sterling). Identifiers cannot include special characters such as !, %, ^, &, *, and . or embedded spaces.

You cannot use a reserved word, such as a Transact-SQL command, as an identifier. For a complete list of reserved words, see Chapter 5, “Reserved Words.”

You cannot use the dash symbol ( – ) as an identifier.

### Short identifiers

The maximum length for these identifiers is 30 bytes:

- Cursor name
- Server name
- Host name
- Login name
- Password
- Host process identification
- Application name
- Initial language name
- Character set name
- User name
- Group name
- Database name
- Logical device name
- Segment name
Identifiers

- Session name
- Execution class name
- Engine name
- Quiesce tag name
- Cache name

Tables beginning with # (temporary tables)

Tables with names that begin with the pound sign (#) are temporary tables. You cannot create other types of objects with names that begin with the pound sign.

Adaptive Server performs special operations on temporary table names to maintain unique naming on a per-session basis. When you create a temporary table with a name of fewer than 238 bytes, the sysobjects name in the tempdb adds 17 bytes to make the table name unique. If the table name is more than 238 bytes, the temporary table name in sysobjects uses only the first 238 bytes, then adds 17 bytes to make it unique.

In versions of Adaptive Server earlier than 15.0, temporary table names in sysobjects were 30 bytes. If you used a table name with fewer than 13 bytes, the name was padded with underscores (_) to 13 bytes, then another 17 bytes of other characters to bring the name up to 30 bytes.

Case sensitivity and identifiers

Sensitivity to the case (upper or lower) of identifiers and data depends on the sort order installed on your Adaptive Server. Case sensitivity can be changed for single-byte character sets by reconfiguring Adaptive Server’s sort order; see the System Administration Guide for more information. Case is significant in utility program options.

If Adaptive Server is installed with a case-insensitive sort order, you cannot create a table named MYTABLE if a table named MyTable or mytable already exists. Similarly, the following command will return rows from MYTABLE, MyTable, or mytable, or any combination of uppercase and lowercase letters in the name:

```
select * from MYTABLE
```
Uniqueness of object names

Object names need not be unique in a database. However, column names and index names must be unique within a table, and other object names must be unique for each owner within a database. Database names must be unique on Adaptive Server.

Using delimited identifiers

Delimited identifiers are object names enclosed in double quotes. Using delimited identifiers allows you to avoid certain restrictions on object names. Table, view, and column names can be delimited by quotes; other object names cannot.

Delimited identifiers can be reserved words, can begin with non-alphabetic characters, and can include characters that would not otherwise be allowed. They cannot exceed 253 bytes.

Warning! Delimited identifiers may not be recognized by all front-end applications and should not be used as parameters to system procedures.

Before creating or referencing a delimited identifier, you must execute:

```sql
set quoted_identifier on
```

Each time you use the delimited identifier in a statement, you must enclose it in double quotes. For example:

```sql
create table "lone"(col1 char(3))
create table "include spaces" (col1 int)
create table "grant"("add" int)
insert "grant"("add") values (3)
```

While the quoted_identifier option is turned on, do not use double quotes around character or date strings; use single quotes instead. Delimiting these strings with double quotes causes Adaptive Server to treat them as identifiers. For example, to insert a character string into col1 of table, use:

```sql
insert "lone"(col1) values ('abc')
```

Do not use:

```sql
insert "lone"(col1) values ("abc")
```

To insert a single quote into a column, use two consecutive single quotation marks. For example, to insert the characters “a’b” into col1 use:
Identifiers

Syntax that includes quotes

When the quoted_identifier option is set to on, you do not need to use double quotes around an identifier if the syntax of the statement requires that a quoted string contain an identifier. For example:

```
set quoted_identifier on
create table 'lone' (c1 int)
```

However, object_id() requires a string, so you must include the table name in quotes to select the information:

```
select object_id('lone')
```

896003192

You can include an embedded double quote in a quoted identifier by doubling the quote:

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

However, there is no need to double the quote when the statement syntax requires the object name to be expressed as a string:

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

Identifying tables or columns by their qualified object name

You can uniquely identify a table or column by adding other names that qualify it—the database name, owner’s name, and (for a column) the table or view name. Each qualifier is separated from the next one by a period. For example:

```
database.OWNER.table_name.column_name
database.OWNER.view_name.column_name
```

The naming conventions are:

```
[[database.]owner.]table_name
[[database.]owner.]view_name
```

Using delimited identifiers within an object name

If you use set quoted_identifier on, you can use double quotes around individual parts of a qualified object name. Use a separate pair of quotes for each qualifier that requires quotes. For example, use:

```
database.OWNER."table_name"."column_name"
```
Do not use:

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

### Omitting the owner name

You can omit the intermediate elements in a name and use dots to indicate their positions, as long as the system is given enough information to identify the object:

```sql
database..table_name
database..view_name
```

### Referencing your own objects in the current database

You need not use the database name or owner name to reference your own objects in the current database. The default value for `owner` is the current user, and the default value for `database` is the current database.

If you reference an object without qualifying it with the database name and owner name, Adaptive Server tries to find the object in the current database among the objects you own.

### Referencing objects owned by the database owner

If you omit the owner name and you do not own an object by that name, Adaptive Server looks for objects of that name owned by the Database Owner. You must qualify objects owned by the Database Owner only if you own an object of the same name, but you want to use the object owned by the Database Owner. However, you must qualify objects owned by other users with the user’s name, whether or not you own objects of the same name.

### Using qualified identifiers consistently

When qualifying a column name and table name in the same statement, be sure to use the same qualifying expressions for each; they are evaluated as strings and must match; otherwise, an error is returned. Example 2 is incorrect because the syntax style for the column name does not match the syntax style used for the table name.

**Example 1**

```sql
select demo.mary.publishers.city
from demo.mary.publishers
```
Identifiers

Example 2

```sql
select demo.mary.publishers.city
from demo..publishers
```

The column prefix "demo.mary.publishers" does not match a table name or alias name used in the query.

Determining whether an identifier is valid

Use the system function `valid_name`, after changing character sets or before creating a table or view, to determine whether the object name is acceptable to Adaptive Server. Here is the syntax:

```sql
select valid_name("Object_name")
```

If `object_name` is not a valid identifier (for example, if it contains illegal characters or is more than 30 bytes long), Adaptive Server returns 0. If `object_name` is a valid identifier, Adaptive Server returns a nonzero number.

Renaming database objects

Rename user objects (including user-defined datatypes) with `sp_rename`.

**Warning!** After you rename a table or column, you must redefine all procedures, triggers, and views that depend on the renamed object.

Using multibyte character sets

In multibyte character sets, a wider range of characters is available for use in identifiers. For example, on a server with the Japanese language installed, the following types of characters may be used as the first character of an identifier: Zenkaku or Hankaku Katakana, Hiragana, Kanji, Romaji, Greek, Cyrillic, or ASCII.
Although Hankaku Katakana characters are legal in identifiers on Japanese systems, they are not recommended for use in heterogeneous systems. These characters cannot be converted between the EUC-JIS and Shift-JIS character sets.

The same is true for some 8-bit European characters. For example, the OE ligature, is part of the Macintosh character set (codepoint 0xCE). This character does not exist in the ISO 8859-1 (iso_1) character set. If the OE ligature exists in data being converted from the Macintosh to the ISO 8859-1 character set, it causes a conversion error.

If an object identifier contains a character that cannot be converted, the client loses direct access to that object.

**Pattern matching with wildcard characters**

Wildcard characters represent one or more characters, or a range of characters, in a *match_string*. A *match_string* is a character string containing the pattern to find in the expression. It can be any combination of constants, variables, and column names or a concatenated expression, such as:

```
like @variable + "%".
```

If the match string is a constant, it must always be enclosed in single or double quotes.

Use wildcard characters with the keyword `like` to find character and date strings that match a particular pattern. You cannot use `like` to search for seconds or milliseconds. For more information, see “Using wildcard characters with datetime data” on page 383.

Use wildcard characters in `where` and `having` clauses to find character or date/time information that is `like`—or not `like`—the match string:

```
(where | having) [not] expression [not] like match_string
[escape "escape_character"]
```

*expression* can be any combination of column names, constants, or functions with a character value.

Wildcard characters used without `like` have no special meaning. For example, this query finds any phone numbers that start with the four characters “415%”:

```
select phone
```
Pattern matching with wildcard characters

from authors
where phone = "415%"

Using not like

Use not like to find strings that do not match a particular pattern. These two queries are equivalent: they find all the phone numbers in the authors table that do not begin with the 415 area code.

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

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

For example, this query finds the system tables in a database whose names begin with "sys":

select name
from sysobjects
where name like "sys%"

To see all the objects that are not system tables, use:

not like "sys%"

If you have a total of 32 objects and like finds 13 names that match the pattern, not like will find the 19 objects that do not match the pattern.

not like and the negative wildcard character [^] may give different results (see “The caret (^) wildcard character” on page 381). You cannot always duplicate not like patterns with like and ^. This is because not like finds the items that do not match the entire like pattern, but like with negative wildcard characters is evaluated one character at a time.

A pattern such as like "[^s][^y][^s]%" may not produce the same results. Instead of 19, you might get only 14, with all the names that begin with “s”, or have “y” as the second letter, or have “s” as the third letter eliminated from the results, as well as the system table names. This is because match strings with negative wildcard characters are evaluated in steps, one character at a time. If the match fails at any point in the evaluation, it is eliminated.
Case and accent insensitivity

If your Adaptive Server uses a case-insensitive sort order, case is ignored when comparing expression and match_string. For example, this clause would return “Smith,” “smith,” and “SMITH” on a case-insensitive Adaptive Server:

    where col_name like "Sm%"

If your Adaptive Server is also accent-insensitive, it treats all accented characters as equal to each other and to their unaccented counterparts, both uppercase and lowercase. The sp_helpsort system procedure displays the characters that are treated as equivalent, displaying an “=” between them.

Using wildcard characters

You can use the match string with a number of wildcard characters, which are discussed in detail in the following sections. Table 4-7 summarizes the wildcard characters:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>%</td>
<td>Any string of 0 or more characters</td>
</tr>
<tr>
<td>_</td>
<td>Any single character</td>
</tr>
<tr>
<td>[ ]</td>
<td>Any single character within the specified range ([a-f]) or set ([abcdef])</td>
</tr>
<tr>
<td>[^]</td>
<td>Any single character not within the specified range ([^a-f]) or set ([^abcdef])</td>
</tr>
</tbody>
</table>

Enclose the wildcard character and the match string in single or double quotes (like “[dD]eFr_nce”).

The percent sign (%) wildcard character

Use the % wildcard character to represent any string of zero or more characters. For example, to find all the phone numbers in the authors table that begin with the 415 area code:

    select phone
    from authors
    where phone like "415%"

To find names that have the characters “en” in them (Bennet, Green, McBadden):

    select au_lname
    from authors
Pattern matching with wildcard characters

where au_lname like "%en%"

Trailing blanks following "%" in a like clause are truncated to a single trailing blank. For example, "%" followed by two spaces matches "X " (one space); "X  " (two spaces); "X   " (three spaces), or any number of trailing spaces.

The underscore (\_) wildcard character

Use the underscore (\_) wildcard character to represent any single character. For example, to find all six-letter names that end with “heryl” (for example, Cheryl):

```
select au_fname
from authors
where au_fname like "_heryl"
```

Bracketed ([ ]) characters

Use brackets to enclose a range of characters, such as [a-f], or a set of characters such as [a2Br]. When ranges are used, all values in the sort order between (and including) rangespec1 and rangespec2 are returned. For example, “[0-z]” matches 0-9, A-Z and a-z (and several punctuation characters) in 7-bit ASCII.

To find names ending with “inger” and beginning with any single character between M and Z:

```
select au_lname
from authors
where au_lname like "[M-Z]inger"
```

To find both “DeFrance” and “deFrance”:

```
select au_lname
from authors
where au_lname like "[dD]eFrance"
```

When using bracketed identifiers to create objects, such as with `create table [table_name]` or `create database [dbname]`, you must include at least one valid character.

All trailing spaces within bracketed identifiers are removed from the object name. For example, you achieve the same results executing the following `create table` commands:

- create table [tab1<space><space>]
- create table [tab1]
• create table [tab1<space><space><space>]
• create table tab1

This rule applies to all objects you can create using bracketed identifiers.

The caret (^) wildcard character

The caret is the negative wildcard character. Use it to find strings that do not match a particular pattern. For example, “[^a-f]” finds strings that are not in the range a-f and “[^a2bR]” finds strings that are not “a,” “2,” “b,” or “R.”

To find names beginning with “M” where the second letter is not “c”:

```
select au_lname
from authors
where au_lname like "M[^c]%"
```

When ranges are used, all values in the sort order between (and including) rangespec1 and rangespec2 are returned. For example, “[0-z]” matches 0-9, A-Z, a-z, and several punctuation characters in 7-bit ASCII.

Using multibyte wildcard characters

If the multibyte character set configured on your Adaptive Server defines equivalent double-byte characters for the wildcard characters _, %, - [], and ^, you can substitute the equivalent character in the match string. The underscore equivalent represents either a single- or double-byte character in the match string.

Using wildcard characters as literal characters

To search for the occurrence of %, _, [, ], or ^ within a string, you must use an escape character. When a wildcard character is used in conjunction with an escape character, Adaptive Server interprets the wildcard character literally, rather than using it to represent other characters.

Adaptive Server provides two types of escape characters:
• Square brackets, a Transact-SQL extension
• Any single character that immediately follows an escape clause, compliant with the SQL standards
**Pattern matching with wildcard characters**

**Using square brackets ([ ]) as escape characters**

Use square brackets as escape characters for the percent sign, the underscore, and the left bracket. The right bracket does not need an escape character; use it by itself. If you use the hyphen as a literal character, it must be the first character inside a set of square brackets.

Table 4-8 shows examples of square brackets used as escape characters with like.

<table>
<thead>
<tr>
<th>like predicate</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>like &quot;5%&quot;</td>
<td>5 followed by any string of 0 or more characters</td>
</tr>
<tr>
<td>like &quot;5[%]&quot;</td>
<td>5%</td>
</tr>
<tr>
<td>like &quot;_.n&quot;</td>
<td>an, in, on (and so on)</td>
</tr>
<tr>
<td>like &quot;[.]n&quot;</td>
<td>_n</td>
</tr>
<tr>
<td>like &quot;[a-cdf]&quot;</td>
<td>a, b, c, d, or f</td>
</tr>
<tr>
<td>like &quot;[-acdf]&quot;</td>
<td>-, a, c, d, or f</td>
</tr>
<tr>
<td>like &quot;[[]]&quot;</td>
<td>[</td>
</tr>
<tr>
<td>like &quot;[]&quot;</td>
<td>]</td>
</tr>
<tr>
<td>like &quot;[][ab]&quot;</td>
<td>[][ab]</td>
</tr>
</tbody>
</table>

**Using the escape clause**

Use the escape clause to specify an escape character. Any single character in the server’s default character set can be used as an escape character. If you try to use more than one character as an escape character, Adaptive Server generates an exception.

Do not use existing wildcard characters as escape characters because:

- If you specify the underscore (_) or percent sign (%) as an escape character, it loses its special meaning within that like predicate and acts only as an escape character.

- If you specify the left or right bracket ([ or ]) as an escape character, the Transact-SQL meaning of the bracket is disabled within that like predicate.

- If you specify the hyphen (-) or caret (^) as an escape character, it loses its special meaning and acts only as an escape character.

An escape character retains its special meaning within square brackets, unlike wildcard characters such as the underscore, the percent sign, and the open bracket.
The escape character is valid only within its `like` predicate and has no effect on other `like` predicates contained in the same statement. The only characters that are valid following an escape character are the wildcard characters ( `_`, `%`, `[`, `]`, or `[^]`), and the escape character itself. The escape character affects only the character following it, and subsequent characters are not affected by it.

If the pattern contains two literal occurrences of the character that happens to be the escape character, the string must contain four consecutive escape characters. If the escape character does not divide the pattern into pieces of one or two characters, Adaptive Server returns an error message. Table 4-9 shows examples of escape clauses used with `like`.

**Table 4-9: Using the escape clause**

<table>
<thead>
<tr>
<th><code>like</code> predicate</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>like &quot;5@%&quot; escape &quot;@&quot;</code></td>
<td>5%</td>
</tr>
<tr>
<td><code>like &quot;_n&quot; escape &quot;\&quot;&quot;</code></td>
<td>_n</td>
</tr>
<tr>
<td><code>like &quot;%80@%&quot; escape &quot;@&quot;</code></td>
<td>String containing 80%</td>
</tr>
<tr>
<td><code>like &quot;_sql&quot;%&quot; escape &quot;\&quot;&quot;</code></td>
<td>String containing _sql%</td>
</tr>
<tr>
<td><code>like &quot;%#####_##%&quot; escape &quot;\&quot;&quot;</code></td>
<td>String containing #_##</td>
</tr>
</tbody>
</table>

**Using wildcard characters with `datetime` data**

When you use `like` with `datetime` values, Adaptive Server converts the dates to the standard `datetime` format, then to `varchar`. Since the standard storage format does not include seconds or milliseconds, you cannot search for seconds or milliseconds with `like` and a pattern.

It is a good idea to use `like` when you search for `datetime` values, since `datetime` entries may contain a variety of date parts. For example, if you insert the value “9:20” and the current date into a column named `arrival_time`, the clause:

    where arrival_time = '9:20'

would not find the value, because Adaptive Server converts the entry into “Jan 1 1900 9:20AM.” However, the following clause would find this value:

    where arrival_time like '%9:20%'
Pattern matching with wildcard characters
Keywords, also known as reserved words, are words that have special meanings. This chapter lists Transact-SQL and ANSI SQL keywords. Topics covered are:

<table>
<thead>
<tr>
<th>Topics</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transact-SQL reserved words</td>
<td>385</td>
</tr>
<tr>
<td>ANSI SQL reserved words</td>
<td>386</td>
</tr>
<tr>
<td>Potential ANSI SQL reserved words</td>
<td>387</td>
</tr>
</tbody>
</table>

**Transact-SQL reserved words**

The words in Table 5-1 are reserved by Adaptive Server as keywords (part of SQL command syntax). They cannot be used as names of database objects such as databases, tables, rules, or defaults. They can be used as names of local variables and as stored procedure parameter names.

To find the names of existing objects that are reserved words, use `sp_checkreswords` in *Reference Manual: Procedures.*

*Table 5-1: List of Transact-SQL reserved words*

<table>
<thead>
<tr>
<th>Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>D</td>
</tr>
<tr>
<td>E</td>
</tr>
<tr>
<td>F</td>
</tr>
<tr>
<td>G</td>
</tr>
<tr>
<td>H</td>
</tr>
</tbody>
</table>
ANSI SQL reserved words

<table>
<thead>
<tr>
<th>Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
</tr>
<tr>
<td>J</td>
</tr>
<tr>
<td>K</td>
</tr>
<tr>
<td>L</td>
</tr>
<tr>
<td>M</td>
</tr>
<tr>
<td>N</td>
</tr>
</tbody>
</table>

Note: Although “new” is not a Transact-SQL reserved word, since it may become a reserved word in the future, Sybase recommends that you avoid using it (for example, to name a database object). “New” is a special case (see “Potential ANSI SQL reserved words” on page 387 for information on other reserved words) because it appears in the spt_values table, and because sp_checkreswords displays “New” as a reserved word.

| O     | of, off, offsets, on, once, online, only, open, option, or, order, out, output, over |
| P     | partition, perm, permanent, plan, prepare, primary, print, privileges, proc, procedure, processexit, proxy_table, public |
| Q     | quiesce |
| R     | raiserror, read, readpast, readtext, reconfigure, references, remove, reorg, replace, replication, reservepagegap, return, returns, revoke, role, rollback, rowcount, rows, rule |
| S     | save, schema, scroll, scrollable, select, semi_sensitive, set, setuser, shared, shutdown, some, statistics, stringsize, stripe, sum, syb_identity, syb_restree, syb_terminate |
| T     | table, temp, temporary, textsize, to, tracefile, tran, transaction, trigger, truncate, tsequal |
| U     | union, unique, unpartition, update, use, user, user_option, using |
| V     | values, varying, view |
| W     | waitfor, when, where, while, with, work, writetext |
| X     | xmlextract, xmlparse, xmltest, xmlvalidate |

ANSI SQL reserved words

Adaptive Server includes entry-level ANSI SQL features. Full ANSI SQL implementation includes the words listed in the following tables as command syntax. Upgrading identifiers can be a complex process; therefore, we are providing this list for your convenience. The publication of this information does not commit Sybase to providing all of these ANSI SQL features in subsequent releases. In addition, subsequent releases may include keywords not included in this list.
The words in Table 5-2 are ANSI SQL keywords that are not reserved words in Transact-SQL.

**Table 5-2: List of ANSI SQL reserved words**

<table>
<thead>
<tr>
<th>Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>absolute, action, allocate, are, assertion</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>bit, bit_length, both</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>cascaded, case, cast, catalog, char, char_length, character, character_length, coalesce, collate, collation, column, connection, constraints, corresponding, cross, current_date, current_time, current_timestamp, current_user</td>
</tr>
<tr>
<td>D</td>
</tr>
<tr>
<td>date, day, dec, decimal, deferrable, deferred, describe, descriptor, diagnostics, disconnect, domain</td>
</tr>
<tr>
<td>E</td>
</tr>
<tr>
<td>end-exec, exception, extract</td>
</tr>
<tr>
<td>F</td>
</tr>
<tr>
<td>false, first, float, found, full</td>
</tr>
<tr>
<td>G</td>
</tr>
<tr>
<td>get, global, go</td>
</tr>
<tr>
<td>H</td>
</tr>
<tr>
<td>hour</td>
</tr>
<tr>
<td>I</td>
</tr>
<tr>
<td>immediate, indicator, initially, inner, input, insensitive, int, integer, interval</td>
</tr>
<tr>
<td>J</td>
</tr>
<tr>
<td>join</td>
</tr>
<tr>
<td>L</td>
</tr>
<tr>
<td>language, last, leading, left, local, lower</td>
</tr>
<tr>
<td>M</td>
</tr>
<tr>
<td>match, minute, module, month</td>
</tr>
<tr>
<td>N</td>
</tr>
<tr>
<td>names, natural, nchar, next, no, nullif, numeric</td>
</tr>
<tr>
<td>O</td>
</tr>
<tr>
<td>octet_length, outer, output, overlaps</td>
</tr>
<tr>
<td>P</td>
</tr>
<tr>
<td>pad, partial, position, preserve, prior</td>
</tr>
<tr>
<td>R</td>
</tr>
<tr>
<td>real, relative, restrict, right</td>
</tr>
<tr>
<td>S</td>
</tr>
<tr>
<td>scroll, second, section, semi_sensitive, session_user, size, smallint, space, sql, sqlcode, sqlerror, sqlstate, substring, system_user</td>
</tr>
<tr>
<td>T</td>
</tr>
<tr>
<td>then, time, timestamp, timezone_hour, timezone_minute, trailing, translate, translation, trim, true</td>
</tr>
<tr>
<td>U</td>
</tr>
<tr>
<td>unknown, upper, usage</td>
</tr>
<tr>
<td>V</td>
</tr>
<tr>
<td>value, varchar</td>
</tr>
<tr>
<td>W</td>
</tr>
<tr>
<td>when, whenever, write, year</td>
</tr>
<tr>
<td>Z</td>
</tr>
<tr>
<td>zone</td>
</tr>
</tbody>
</table>

**Potential ANSI SQL reserved words**

If you are using the ISO/IEC 9075:1989 standard, also avoid using the words shown in the following list because these words may become ANSI SQL reserved words in the future.
### Potential ANSI SQL reserved words

<table>
<thead>
<tr>
<th>Words</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>after, alias, async</td>
</tr>
<tr>
<td>B</td>
<td>before, boolean, breadth</td>
</tr>
<tr>
<td>C</td>
<td>call, completion, cycle</td>
</tr>
<tr>
<td>D</td>
<td>data, depth, dictionary</td>
</tr>
<tr>
<td>E</td>
<td>each, elseif, equals</td>
</tr>
<tr>
<td>G</td>
<td>general</td>
</tr>
<tr>
<td>I</td>
<td>ignore</td>
</tr>
<tr>
<td>L</td>
<td>leave, less, limit, loop</td>
</tr>
<tr>
<td>M</td>
<td>modify</td>
</tr>
<tr>
<td>N</td>
<td>new, none</td>
</tr>
<tr>
<td>O</td>
<td>object, oid, old, operation, operators, others</td>
</tr>
<tr>
<td>P</td>
<td>parameters, pendant, preorder, private, protected</td>
</tr>
<tr>
<td>R</td>
<td>recursive, ref, referencing, resignal, return, returns, routine, row</td>
</tr>
<tr>
<td>S</td>
<td>savepoint, search, sensitive, sequence, signal, similar, sqlexception, structure</td>
</tr>
<tr>
<td>T</td>
<td>test, there, type</td>
</tr>
<tr>
<td>U</td>
<td>under</td>
</tr>
<tr>
<td>V</td>
<td>variable, virtual, visible</td>
</tr>
<tr>
<td>W</td>
<td>wait, without</td>
</tr>
</tbody>
</table>
CHAPTER 6

SQLSTATE Codes and Messages

This chapter describes Adaptive Server’s SQLSTATE status codes and their associated messages.

Topics covered are:

<table>
<thead>
<tr>
<th>Topics</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warnings</td>
<td>389</td>
</tr>
<tr>
<td>Exceptions</td>
<td>390</td>
</tr>
</tbody>
</table>

SQLSTATE codes are required for entry level ANSI SQL compliance. They provide diagnostic information about two types of conditions:

- **Warnings** – conditions that require user notification but are not serious enough to prevent a SQL statement from executing successfully
- **Exceptions** – conditions that prevent a SQL statement from having any effect on the database

Each SQLSTATE code consists of a 2-character class followed by a 3-character subclass. The class specifies general information about error type. The subclass specifies more specific information.

SQLSTATE codes are stored in the `sysmessages` system table, along with the messages that display when these conditions are detected. Not all Adaptive Server error conditions are associated with a SQLSTATE code—only those mandated by ANSI SQL. In some cases, multiple Adaptive Server error conditions are associated with a single SQLSTATE value.

## Warnings

Adaptive Server currently detects the following SQLSTATE warning conditions, described in Table 6-1:
Exceptions

Table 6-1: SQLSTATE warnings

<table>
<thead>
<tr>
<th>Message</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warning – null value eliminated in set function.</td>
<td>01003</td>
<td>Occurs when you use an aggregate function (avg, max, min, sum, or count) on an expression with a null value.</td>
</tr>
<tr>
<td>Warning–string data, right truncation</td>
<td>01004</td>
<td>Occurs when character, unichar, or binary data is truncated to 255 bytes. The data may be:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The result of a select statement in which the client does not support the WIDE TABLES property.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Parameters to an RPC on remote Adaptive Servers or Open Servers that do not support the WIDE TABLES property.</td>
</tr>
</tbody>
</table>

Exceptions

Adaptive Server detects the following types of exceptions:

- Cardinality violations
- Data exceptions
- Integrity constraint violations
- Invalid cursor states
- Syntax errors and access rule violations
- Transaction rollbacks
- with check option violations

Exception conditions are described in Table 6-2 through Table 6-8. Each class of exceptions appears in its own table. Within each table, conditions are sorted alphabetically by message text.

Cardinality violations

Cardinality violations occur when a query that should return only a single row returns more than one row to an Embedded SQL™ application.
Table 6-2: Cardinality violations

<table>
<thead>
<tr>
<th>Message</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
</table>
| Subquery returned more than 1 value. This is illegal when the subquery follows =, !=, <, <=, >, >=, or when the subquery is used as an expression. | 21000 | Occurs when:  
  - A scalar subquery or a row subquery returns more than one row.  
  - A `select into parameter_list` query in Embedded SQL returns more than one row. |

Data exceptions

Data exceptions occur when an entry:

- Is too long for its datatype,
- Contains an illegal escape sequence, or
- Contains other format errors.

Table 6-3: Data exceptions

<table>
<thead>
<tr>
<th>Message</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
</table>
| Arithmetic overflow occurred.                                          | 22003 | Occurs when:  
  - An exact numeric type would lose precision or scale as a result of an arithmetic operation or `sum` function.  
  - An approximate numeric type would lose precision or scale as a result of truncation, rounding, or a `sum` function. |
| Data exception - string data right truncated.                          | 22001 | Occurs when a `char`, `unichar`, `univarchar`, or `varchar` column is too short for the data being inserted or updated and non-blank characters must be truncated. |
| Divide by zero occurred.                                               | 22012 | Occurs when a numeric expression is being evaluated and the value of the divisor is zero. |
| Illegal escape character found.                                         | 22019 | Occurs when you are searching for strings that match a given pattern if the escape sequence does not consist of a single character. |
| Invalid pattern string. The character following the escape character must be percent sign, underscore, left square bracket, right square bracket, or the escape character. | 22025 | Occurs when you are searching for strings that match a particular pattern when:  
  - The escape character is not immediately followed by a percent sign, an underscore, or the escape character itself, or  
  - The escape character partitions the pattern into substrings whose lengths are other than 1 or 2 characters. |
Exceptions

Integrity constraint violations

Integrity constraint violations occur when an insert, update, or delete statement violates a primary key, foreign key, check, or unique constraint or a unique index.

Table 6-4: Integrity constraint violations

<table>
<thead>
<tr>
<th>Message</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attempt to insert duplicate key row in object object_name with unique index index_name.</td>
<td>23000</td>
<td>Occurs when a duplicate row is inserted into a table that has a unique constraint or index.</td>
</tr>
<tr>
<td>Check constraint violation occurred, dbname = database_name, table name = table_name, constraint name = constraint_name.</td>
<td>23000</td>
<td>Occurs when an update or delete would violate a check constraint on a column.</td>
</tr>
<tr>
<td>Dependent foreign key constraint violation in a referential integrity constraint. dbname = database_name, table name = table_name, constraint name = constraint_name.</td>
<td>23000</td>
<td>Occurs when an update or delete on a primary key table would violate a foreign key constraint.</td>
</tr>
<tr>
<td>Foreign key constraint violation occurred, dbname = database_name, table name = table_name, constraint name = constraint_name.</td>
<td>23000</td>
<td>Occurs when an insert or update on a foreign key table is performed without a matching value in the primary key table.</td>
</tr>
</tbody>
</table>

Invalid cursor states

Invalid cursor states occur when:

- A fetch uses a cursor that is not currently open, or
- An update where current of or delete where current of affects a cursor row that has been modified or deleted, or
- An update where current of or delete where current of affects a cursor row that not been fetched.

Table 6-5: Invalid cursor states

<table>
<thead>
<tr>
<th>Message</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attempt to use cursor cursor_name which is not open. Use the system stored procedure sp_cursorinfo for more information.</td>
<td>24000</td>
<td>Occurs when an attempt is made to fetch from a cursor that has never been opened or that was closed by a commit statement or an implicit or explicit rollback. Reopen the cursor and repeat the fetch.</td>
</tr>
</tbody>
</table>
Syntax errors and access rule violations

Syntax errors are generated by SQL statements that contain unterminated comments, implicit datatype conversions not supported by Adaptive Server or other incorrect syntax.

Access rule violations are generated when a user tries to access an object that does not exist or one for which he or she does not have the correct permissions.

Table 6-6: Syntax errors and access rule violations

<table>
<thead>
<tr>
<th>Message</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command permission denied on object object_name, database database_name, owner owner_name.</td>
<td>42000</td>
<td>Occurs when a user tries to access an object for which he or she does not have the proper permissions.</td>
</tr>
<tr>
<td>Implicit conversion from datatype 'datatype' to 'datatype' is not allowed. Use the CONVERT function to run this query.</td>
<td>42000</td>
<td>Occurs when the user attempts to convert one datatype to another but Adaptive Server cannot do the conversion implicitly.</td>
</tr>
<tr>
<td>Incorrect syntax near object_name.</td>
<td>42000</td>
<td>Occurs when incorrect SQL syntax is found near the object specified.</td>
</tr>
</tbody>
</table>
**Exceptions**

<table>
<thead>
<tr>
<th>Message</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert error: column name or number of supplied values does not match table definition.</td>
<td>42000</td>
<td>Occurs during inserts when an invalid column name is used or when an incorrect number of values is inserted.</td>
</tr>
<tr>
<td>Missing end comment mark <code>*/</code>.</td>
<td>42000</td>
<td>Occurs when a comment that begins with the <code>/*</code> opening delimiter does not also have the <code>*/</code> closing delimiter.</td>
</tr>
<tr>
<td>object_name not found. Specify owner.objectname or use sp_help to check whether the object exists (sp_help may produce lots of output).</td>
<td>42000</td>
<td>Occurs when a user tries to reference an object that he or she does not own. When referencing an object owned by another user, be sure to qualify the object name with the name of its owner.</td>
</tr>
</tbody>
</table>
| The size (size) given to the object_name exceeds the maximum. The largest size allowed is size. | 42000 | Occurs when:  
  - The total size of all the columns in a table definition exceeds the maximum allowed row size.  
  - The size of a single column or parameter exceeds the maximum allowed for its datatype. |

**Transaction rollbacks**

Transaction rollbacks occur when the transaction isolation level is set to 3, but Adaptive Server cannot guarantee that concurrent transactions can be serialized. This type of exception generally results from system problems such as disk crashes and offline disks.

**Table 6-7: Transaction rollbacks**

<table>
<thead>
<tr>
<th>Message</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Your server command (process id #process_id ) was deadlocked with another process and has been chosen as deadlock victim. Re-run your command.</td>
<td>40001</td>
<td>Occurs when Adaptive Server detects that it cannot guarantee that two or more concurrent transactions can be serialized.</td>
</tr>
</tbody>
</table>

**with check option violation**

This class of exception occurs when data being inserted or updated through a view would not be visible through the view.
### Table 6-8: with check option violation

<table>
<thead>
<tr>
<th>Message</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>The attempted insert or update failed because the target view was either created WITH CHECK OPTION or spans another view created WITH CHECK OPTION. At least one resultant row from the command would not qualify under the CHECK OPTION constraint.</td>
<td>44000</td>
<td>Occurs when a view, or any view on which it depends, was created with a with check option clause.</td>
</tr>
</tbody>
</table>
Symbols

& (ampersand) “and” bitwise operator 362
* (asterisk)
  for overlength numbers 289
  multiplication operator 361
\ (backslash) character string continuation with 369
 := (BNF notation)
  in SQL statements xx
^ (caret)
  “exclusive or” bitwise operator 362
  wildcard character 379, 381
: (colon) preceding milliseconds 74, 151
, (comma)
  in default print format for money values 18
  not allowed in money values 19
  in SQL statements xx
{ } (curly braces)
  in SQL statements xx
$ (dollar sign)
  in identifiers 371
  in money datatypes 19
.. (dots) in database object names 375
|| (double pipe)
  string concatenation operator 363
= (equals sign) comparison operator 364
> (greater than) comparison operator 364
>= (greater than or equal to) comparison operator 364
< (less than) comparison operator 364
<= (less than or equal to) comparison operator 364
- (minus sign)
  arithmetic operator 361
  for negative monetary values 19
  in integer data 13
!= (not equal to) comparison operator 364
<> (not equal to) comparison operator 364
!= (not greater than) comparison operator 364
!$ (not less than) comparison operator 364
() (parentheses)
  in expressions 368
  in SQL statements xx
% (percent sign)
  arithmetic operator (modulo) 361
  wildcard character 379
. (period)
  preceding milliseconds 74, 151
  separator for qualifier names 374
| (pipe) “or” bitwise operator 362
+ (plus)
  arithmetic operator 361
  in integer data 13
  null values and 364
  string concatenation operator 363
£ (pound sterling sign)
  in identifiers 371
  in money datatypes 19
“ ” (quotation marks)
  in identifiers 371
  in SQL statements xx
  enclosing constant values 76
  enclosing datetime values 22
  enclosing empty strings 367, 369
  in expressions 369
  literal specification of 369
/ (slash) arithmetic operator (division) 361
[ ] (square brackets)
  character set wildcard 379, 380
  in SQL statements xx
[^] (square brackets and caret) character set wildcard 379
~ (tilde) “not” bitwise operator 362
_ (underscore)
  object identifier prefix 324, 370
  in temporary table names 372
  character string wildcard 379, 380
¥ (yen sign)
  in identifiers 371
  in money datatypes 19
@@cursor_rows global variable 353
Index

Numerics
21st century numbers 22

A
abbreviations
chars for characters, patindex 230, 235
date parts 73, 150
abort option, set_let_admin function 201
abs mathematical function 80
accent sensitivity, wildcard characters and 379
ACF. See Application Context Facility
acos mathematical function 81
adding
interval to a date 142
timestamp column 314
user-defined datatypes 45
addition operator (+) 361
aggregate functions 54–60
See also row aggregates; individual function names
avg 87
count 126
count_big 128–129
difference from row aggregates 58
group by clause and 54, 56
having clause and 54
max 213
min 215
scalar aggregates 55
sum 299
vector aggregates 55
aggregate functions and cursors 57
all keyword including subqueries 365
alter table command, adding timestamp column 314
ampersand (&) “and” bitwise operator 362
and (&) bitwise operator 362
and keyword
in expressions 367
range-end 365
angles, mathematical functions for 81
ANSI SQL datatypes 11
any keyword in expressions 365
application attributes 266
Application Context Facility (ACF) 266
application contexts
getting 172
listing 207
removing 257
setting 266
approximate numeric datatypes 16
arithabort option, set
arith_overflow and 11, 70
mathematical functions and arith_overflow 75
mathematical functions and numeric_truncation 71, 75
arithignore option, set
arith_overflow and 70
mathematical functions and arith_overflow 75
arithmetic
errors 75
expressions 360
operations, approximate numeric datatypes and 16
operations, exact numeric datatypes and 13
operations, money datatypes and 18
operators, in expressions 361
ASCII characters 82
ascii string function 82
asehostname function 83
asin mathematical function 84
asterisk (*)
multiplication operator 361
overlength numbers 289
atan mathematical function 85
@@authmech global variable 351
@@bootcount global variable 351
@@boottime global variable 351
@@bulkarraysize global variable 351
@@bulkbatchsize global variable 352
@@char_convert global variable 352
@@cis_rpc_handling global variable 352
@@cis_version global variable 352
@@client_csexpansion global variable 352
@@client_csname global variable 352
@@client_csname global variable 352
@@cmpstate global variable 352
@@connections global variable 352
@@cpu_busy global variable 352
@@curloid global variable 353
@@datefirst global variable 353
@@dbs global variable 353
@@error global variable 353
<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>@@errorlog</td>
<td>Global Variable</td>
<td>353</td>
</tr>
<tr>
<td>@@failedoverconn</td>
<td>Global Variable</td>
<td>353</td>
</tr>
<tr>
<td>@@fetch_status</td>
<td>Global Variable</td>
<td>353</td>
</tr>
<tr>
<td>@@guestuserid</td>
<td>Global Variable</td>
<td>353</td>
</tr>
<tr>
<td>@@hacmpservername</td>
<td>Global Variable</td>
<td>353</td>
</tr>
<tr>
<td>@@haconnection</td>
<td>Global Variable</td>
<td>353</td>
</tr>
<tr>
<td>@@heapm.getSize</td>
<td>Global Variable</td>
<td>353</td>
</tr>
<tr>
<td>@@identity</td>
<td>Global Variable</td>
<td>353</td>
</tr>
<tr>
<td>@@idle</td>
<td>Global Variable</td>
<td>354</td>
</tr>
<tr>
<td>@@invaliduserid</td>
<td>Global Variable</td>
<td>354</td>
</tr>
<tr>
<td>@@io_busy</td>
<td>Global Variable</td>
<td>354</td>
</tr>
<tr>
<td>@@isolation</td>
<td>Global Variable</td>
<td>354</td>
</tr>
<tr>
<td>@@kernel_addr</td>
<td>Global Variable</td>
<td>354</td>
</tr>
<tr>
<td>@@kernel_size</td>
<td>Global Variable</td>
<td>354</td>
</tr>
<tr>
<td>@@langid</td>
<td>Global Variable</td>
<td>354</td>
</tr>
<tr>
<td>@@language</td>
<td>Global Variable</td>
<td>354</td>
</tr>
<tr>
<td>@@lastlogindate</td>
<td>Global Variable</td>
<td>354</td>
</tr>
<tr>
<td>@@lock_timeout</td>
<td>Global Variable</td>
<td>354</td>
</tr>
<tr>
<td>@@max_connections</td>
<td>Global Variable</td>
<td>354</td>
</tr>
<tr>
<td>@@max_precision</td>
<td>Global Variable</td>
<td>354</td>
</tr>
<tr>
<td>@@maxcharlen</td>
<td>Global Variable</td>
<td>354</td>
</tr>
<tr>
<td>@@maxgroupid</td>
<td>Global Variable</td>
<td>354</td>
</tr>
<tr>
<td>@@maxpagesize</td>
<td>Global Variable</td>
<td>354</td>
</tr>
<tr>
<td>@@maxspid</td>
<td>Global Variable</td>
<td>354</td>
</tr>
<tr>
<td>@@maxsuid</td>
<td>Global Variable</td>
<td>354</td>
</tr>
<tr>
<td>@@maxuserid</td>
<td>Global Variable</td>
<td>354</td>
</tr>
<tr>
<td>@@mempool_addr</td>
<td>Global Variable</td>
<td>354</td>
</tr>
<tr>
<td>@@min_poolsize</td>
<td>Global Variable</td>
<td>355</td>
</tr>
<tr>
<td>@@mingroupid</td>
<td>Global Variable</td>
<td>355</td>
</tr>
<tr>
<td>@@minspid</td>
<td>Global Variable</td>
<td>355</td>
</tr>
<tr>
<td>@@minsuid</td>
<td>Global Variable</td>
<td>355</td>
</tr>
<tr>
<td>@@minuserid</td>
<td>Global Variable</td>
<td>355</td>
</tr>
<tr>
<td>@@monitors_active</td>
<td>Global Variable</td>
<td>355</td>
</tr>
<tr>
<td>@@ncharsize</td>
<td>Global Variable</td>
<td>355</td>
</tr>
<tr>
<td>@@nestlevel</td>
<td>Global Variable</td>
<td>355</td>
</tr>
<tr>
<td>@@nodeid</td>
<td>Global Variable</td>
<td>355</td>
</tr>
<tr>
<td>@@optgoal</td>
<td>Global Variable</td>
<td>355</td>
</tr>
<tr>
<td>@@options</td>
<td>Global Variable</td>
<td>355</td>
</tr>
<tr>
<td>@@opttimeout</td>
<td>Global Variable</td>
<td>355</td>
</tr>
<tr>
<td>@@pack_received</td>
<td>Global Variable</td>
<td>355</td>
</tr>
<tr>
<td>@@pack_sent</td>
<td>Global Variable</td>
<td>355</td>
</tr>
<tr>
<td>@@packet_errors</td>
<td>Global Variable</td>
<td>355</td>
</tr>
<tr>
<td>@@pagesize</td>
<td>Global Variable</td>
<td>355</td>
</tr>
<tr>
<td>@@parallel_degree</td>
<td>Global Variable</td>
<td>355</td>
</tr>
<tr>
<td>@@probesuid</td>
<td>Global Variable</td>
<td>355</td>
</tr>
<tr>
<td>@@procid</td>
<td>Global Variable</td>
<td>355</td>
</tr>
<tr>
<td>@@recovery_state</td>
<td>Global Variable</td>
<td>355</td>
</tr>
<tr>
<td>@@repartition_degree</td>
<td>Global Variable</td>
<td>355</td>
</tr>
<tr>
<td>@@resource_granularity</td>
<td>Global Variable</td>
<td>356</td>
</tr>
<tr>
<td>@@rowcount</td>
<td>Global Variable</td>
<td>356</td>
</tr>
<tr>
<td>@@scan_parallel_degree</td>
<td>Global Variable</td>
<td>356</td>
</tr>
<tr>
<td>@@servername</td>
<td>Global Variable</td>
<td>356</td>
</tr>
<tr>
<td>@@setrowcount</td>
<td>Global Variable</td>
<td>356</td>
</tr>
<tr>
<td>@@shmem_flags</td>
<td>Global Variable</td>
<td>356</td>
</tr>
<tr>
<td>@@spid</td>
<td>Global Variable</td>
<td>356</td>
</tr>
<tr>
<td>@@sqlstatus</td>
<td>Global Variable</td>
<td>356</td>
</tr>
<tr>
<td>@@ssl_ciphersuite</td>
<td>Global Variable</td>
<td>356</td>
</tr>
<tr>
<td>@@stringsize</td>
<td>Global Variable</td>
<td>356</td>
</tr>
<tr>
<td>@@tempdbid</td>
<td>Global Variable</td>
<td>356</td>
</tr>
<tr>
<td>@@textcolid</td>
<td>Global Variable</td>
<td>41, 356</td>
</tr>
<tr>
<td>@@textdataptnid</td>
<td>Global Variable</td>
<td>357</td>
</tr>
<tr>
<td>@@textdbid</td>
<td>Global Variable</td>
<td>41, 357</td>
</tr>
<tr>
<td>@@textobjid</td>
<td>Global Variable</td>
<td>357</td>
</tr>
<tr>
<td>@@textptnid</td>
<td>Global Variable</td>
<td>357</td>
</tr>
<tr>
<td>@@textptr</td>
<td>Global Variable</td>
<td>41, 357</td>
</tr>
<tr>
<td>@@textptr_parameters</td>
<td>Global Variable</td>
<td>357</td>
</tr>
<tr>
<td>@@textsize</td>
<td>Global Variable</td>
<td>41, 357</td>
</tr>
<tr>
<td>@@textts</td>
<td>Global Variable</td>
<td>41, 357</td>
</tr>
<tr>
<td>@@thresh_hysteresis</td>
<td>Global Variable</td>
<td>357</td>
</tr>
<tr>
<td>@@timeticks</td>
<td>Global Variable</td>
<td>357</td>
</tr>
<tr>
<td>@@total_errors</td>
<td>Global Variable</td>
<td>357</td>
</tr>
<tr>
<td>@@total_read</td>
<td>Global Variable</td>
<td>357</td>
</tr>
<tr>
<td>@@total_write</td>
<td>Global Variable</td>
<td>357</td>
</tr>
<tr>
<td>@@tranchained</td>
<td>Global Variable</td>
<td>357</td>
</tr>
<tr>
<td>@@trancount</td>
<td>Global Variable</td>
<td>357</td>
</tr>
<tr>
<td>@@transactional_rpc</td>
<td>Global Variable</td>
<td>357</td>
</tr>
<tr>
<td>@@transtate</td>
<td>Global Variable</td>
<td>357</td>
</tr>
<tr>
<td>@@unicharsize</td>
<td>Global Variable</td>
<td>357</td>
</tr>
<tr>
<td>@@version</td>
<td>Global Variable</td>
<td>357</td>
</tr>
<tr>
<td>@@version_as_integer</td>
<td>Global Variable</td>
<td>357</td>
</tr>
<tr>
<td>@@version_number</td>
<td>Global Variable</td>
<td>357</td>
</tr>
<tr>
<td>atn2</td>
<td>Mathematical Function</td>
<td>86</td>
</tr>
<tr>
<td>auditing</td>
<td>Attributes, setting in an application</td>
<td>266</td>
</tr>
<tr>
<td>audit_event_name</td>
<td>Function</td>
<td>89</td>
</tr>
<tr>
<td>auditing</td>
<td>audity_event_name</td>
<td>89</td>
</tr>
<tr>
<td>audit_event_name</td>
<td>Function</td>
<td>89</td>
</tr>
<tr>
<td>automatic operations, updating columns with timestamp</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>avg</td>
<td>Aggregate Function</td>
<td>87</td>
</tr>
</tbody>
</table>

Reference Manual: Building Blocks
Index

B
backslash (\) for character string continuation 369
Backus Naur Form (BNF) notation xix, xx
base 10 logarithm function 210
between keyword 365
bigint datatype 13
biginttohex datatype conversion function 92
binary
datatypes 32–34
datatypes, trailing zeros in 33
expressions 359
expressions, concatenating 363
representation of data for bitwise operations 362
sort 116, 275
binary datatype 32–35
bintofstr function 93
bit datatype 35
bitwise operators 362–363
blanks
See also spaces, character
class datatypes and 28–31
comparisons 364
empty string evaluated as 369
like and 380
removing leading, with ltrim function 212
removing trailing, with rtrim function 264
BNF notation in SQL statements xix, xx
boolean (logical) expressions 359
@@bootcount global variable 351
@@boottime global variable 351
brackets. See square brackets [ ]
browse mode and timestamp datatype 19, 313
built-in function, ACF 266
built-in functions 47–325
See also individual function names
aggregate 54
conversion 63
date 73
image 78
mathematical 74
security 75
string 76
system 77
text 78
type conversion 118–123
@@bulkarraysize global variable 351
@@bulkbatchsize global variable 352
by row aggregate subgroup 58
C
cache_usagedefault para font> function 95
calculating dates 145
caldayofweek date part 150
calweekofyear date part 150
calyearofweek date part 150
case expressions 96–98, 221–222
null values and 97, 109, 221
case sensitivity
comparison expressions and 364, 379
identifiers and 372
in SQL xx

cast function 99–101
cdw. See caldayofweek date part

cel\_leding mathematical function 102
chained parts of text or image data 37
char datatype 27–29
in expressions 368
char string function 104
@@char_convert global variable 352
char\_length string function 106
character data, avoiding “NULL” in 367
character datatypes 27–32
character expressions
blanks or spaces in 28–31
defined 359
syntax 360
character sets
conversion errors 377
iso_1 377
multibyte 376
object identifiers and 376
character strings
continuation with backslash (\) 369
empty 369
specifying quotes within 369
wildcards in 377
characters
See also spaces, character
0x 71
deleting, using stuff function 296
number of 106
wildcard 377–383

charindex string function 108
@@cis_rpc_handling global variable 352
@@cis_version global variable 352
client, host computer name and 184
@@client_csexpansion global variable 352
@@client_csid global variable 352
@@client_csnam global variable 352
@@cmpstate global variable 352
coalesce function 109–110
coalessce keyword, case 109
codes, soundex 277
col_length system function 111
col_name system function 112
colon (:), preceding milliseconds 151
column identifiers. See identifiers. column name
as qualifier 374
in parentheses 58
returning 112
columns
identifying 374
length definition 111
length of 111
numeric, and row aggregates 58
sizes of (list) 2
comma (,)
default print format for money values 18
not allowed in money values 19
in SQL statements xx
compare system function 113
comparing values
difference string function 168
in expressions 364
timestamp 313
comparison operators
See also relational expressions
in expressions 364
symbols for 364
compute clause and row aggregates 57
computing dates 145
concatenation
null values 364
using + operator 363
using || operator 363

@@connections global variable 352

costants
and string functions 76
comparing in expressions 364
expression for 359
string functions and 76

continuation lines, character string 369

currencies
See also syntax
transaction identifier name 374
Transact-SQL syntax xix
used in the Reference Manual xix
conversion
automatic values 9
between character sets 377
character value to ASCII code 82
dates used with like keyword 26
degrees to radians 243
implicit 9, 368
integer value to character value 104, 310
lower to higher datatypes 368
lowercase to uppercase 315, 316, 317, 318
null values and automatic 10
radians to degrees 162
string concatenation 363
styles for dates 119
upcase to lowercase 211
convert datatype conversion function 118
concatenation and 363
date styles 119
converting hexadecimal numbers 71


cos mathematical function 124
cot mathematical function 125


count aggregate function 126
count_big aggregate function 128–129


CP 850 Alternative
lower case first 116, 275
no accent 116, 275
no case preference 116, 275


CP 850 Scandinavian
dictionary 116, 275


@@cpu_busy global variable 352

create table command and null values 367
@@curloid global variable 353
curly braces (()) in SQL statements xx
currency symbols 19, 371
D

data_pages system function 136–137

database object owners and identifiers 375
database objects

See also individual object names

ID number 227

identifier names 369

user-defined datatypes as 45
database owners

name as qualifier 374, 375

objects and identifiers 375
databases

See also database objects

gaining name of 160

ID number, db_id function 158
datachange system function 138–139
datalength system function 140

compared to col_length 111
datatype conversions

biginttohex 92

binary and numeric data 72

bit information 72

character information 67

convert function 118, 121
date and time information 69
domain errors 71, 100, 121

functions for 63–72

hexadecimal-like information 71

hextobigint 181

hextoint 182

hextoint function 181, 182

image 72, 100, 122

implicit 63

inttohex 190

money information 68

numeric information 68, 69

overflow errors 70

rounding during 68

scale errors 70
datatype precedence. See precedence

datatypes 1–45

See also user-defined datatypes; individual datatype names

ANSI SQL 11

approximate numeric 16

binary 32–34

bit 35

date and time 20–26

datetime values comparison 364

decimal 14–15

dropping user-defined 45

exact numeric 12–15

hierarchy 7

integer 13–14

mixed, arithmetic operations on 361

summary of 2–4

synonyms for 2

trailing zeros in binary 33

Transact-SQL extensions 11

user-defined 11

varbinary 273

date and time datatype 22–26

date datatype 21

date functions 73–74

See also individual function names

current_date 130, 131, 132

current_time 133

dateadd 141

datediff 144

datename 148

datepart 150

day 155

getdate 173

month 216

year 350

date parts
abbreviation names and values 73, 150
caldayofweek 150
calweekofyear 150
calyearofweek 150
entering order of 23
dateadd date function 141
datediff date function 144
datediff function 145–146
datefirst option, set 149, 153
dateformat option, set 23
datename date function 148
datepart date function 150
dates comparing 364
datatypes 20–26
default display settings 24
display formats 20
earliest allowed 22, 73, 143
entry formats 23
pre-1753 datatypes for 73, 143
datetime datatype 22–26
comparison of 364
conversion 26
date functions and 151
values and comparisons 26
day date function 155
day date part 73, 150
dayofyear date part abbreviation and values 73, 150
db_id system function 158, 160
db_name system function 160
db_recovery_status function 161
DB-Library programs, overflow errors in 88, 300
@@dbts global variable 353
dd. See day date part.
decimal datatype 14–15
decimal numbers
round function and 261
str function, representation of 289
decimal points
datatypes, allowing in 14
in integer data 13
default settings
date display format 20, 24
weekday order 153
default values
datatype length 118
datatype precision 118
datatype scale 118
degrees mathematical function 162
degrees, conversion to radians 243
delete command and text row 40
derived_stat system function 163
devices. See sysdevices table.
difference string function 168
division operator (/) 361
dollar sign ($) in identifiers 371
in money datatypes 19
domain rules, mathematical functions errors in 75
dots (..) for omitted name elements 375
double pipe (||)
string concatenation operator 363
double precision datatype 17
double-byte characters. See Multibyte character sets.
double-precision floating-point values 17
doubling quotes in expressions 369
in character strings 29
dropping character with stuff function 296
leading or trailing blanks 212
duplicate rows, text or image 43
duplication of text. See replicate string function
dw. See weekday date part.
dy. See dayofyear date part.

E
e or E exponent notation
approximate numeric datatypes 17
float datatype 6
money datatypes 19
embedded spaces. See spaces, character.
empty string (" ") or ( ’ ’) not evaluated as null 367
as a single space 31, 369
enclosing quotes in expressions 369
equal to. See comparison operators
@@error global variable 353
error handling, domain or range 75
Index

@@errorlog global variable 353
errors
  arithmetic overflow 70
  cast function 100
  convert function 67–71, 121
divide-by-zero 70
domain 71, 100, 121
  scale 70
  trapping mathematical 75
escape characters 382
escape keyword 382–383
european characters in object identifiers 377
exact numeric datatypes 12–15
  arithmetic operations and 13
exists keyword in expressions 365
exp mathematical function 169
explicit null value 367
exponent, datatype (e or E)
  approximate numeric types 17
  float datatype 6
  money types 19
exponential value 169
expressions
  defined 359
  enclosing quotes in 369
  including null values 365
  name and table name qualifying 375
  types of 359

F

@@failedoverconn global variable 353
@@fetch_status global variable 353
finding
  database ID 158
  database name 160
  server user ID 301
  server user name 302, 303, 313, 319
  starting position of an expression 108
  user aliases 325
  user IDs 322
  user names 321, 323
  valid identifiers 324
first-of-the-months, number of 146
fixed-length columns

binary datatypes for 32
character datatypes for 28
null values in 10
float datatype 17
floating-point data 359
str character representation of 289
floor mathematical function 170, 171
formats, date. See dates.
free pages, curunreservedpgs system function 135
front-end applications, browse mode and 313
functions 47
  abs mathematical function 80
  acos mathematical function 81
  aggregate 54
  ascii string function 82
  asehostname function 83
  asin mathematical function 84
  atan mathematical function 85
  atn2 mathematical function 86
  avg aggregate function 87
  biginttohex datatype conversion function 92
  bintostr 93
  cache_usage 95
  cast function 99–101
  ceiling mathematical function 102
  char string function 104
  char_length string function 106
  charindex string function 108
  coalesce function 109–110
  col_length system function 111
  col_name system function 112
  compare system function 113
  conversion 63
  convert datatype conversion function 118
  cos mathematical function 124
  cot mathematical function 125
  count aggregate function 126
  count_big aggregate function 128–129
  current_date date function 130, 131, 132
  current_time date function 133
  curunreservedpgs system function 134
  data_pages system function 136–137
  datachange system function 138–139
  datalength system function 140
  date 73
dataadd date function 141
Index

datediff date function 144
datename date function 148
datepart date function 150
day date function 155
db_id system function 158, 160
db_recovery_status 161
degrees mathematical function 162
derived_stat system function 163
difference string function 168
exp mathematical function 169
floor mathematical function 170
get_appcontext security function 172
gedate date function 173
has_role system function 175
hash system function 177
hextobigint datatype conversion function 181
hextoint datatype conversion function 182
host_id system function 183
host_name system function 184
image 78
index_col system function 187
index_colorder system function 188
instance_name 198
intohex datatype conversion function 190
is_quiesced function 193–194
is_sec_service_on security function 195
isnull function 196
isnumeric 197
lc_id 199
lc_name 200
lct_admin system function 201
len string function 205
license_enabled system function 206
list_appcontext security function 207
lockscheme system function 208
log mathematical function 209
log10 mathematical function 210
lower string function 211
ltrim string function 212
mathematical 74
max aggregate function 213
min aggregate function 215
month date function 216
mut_excl_roles system function 217
newid system function 218
next_identity system function 220
object_id system function 227
object_name system function 228
object_owner_id 229
pagesize system function 230
partition_id 232
partition_id system function 232
partition_name 233
partition_name system function 233
partition_object_id 234
partition_object_id system function 234
patindex string function 235
pi mathematical function 238
power mathematical function 239
proc_role system function 240
pssinfo 242
pssinfo system function 242
radians mathematical function 243
rand mathematical function 244, 245
replicate string function 246
reserve_identity function 247
reserved_pages system function 250
reverse string function 254
right string function 255
rm_appcontext security function 257
role_contain system function 258
role_id system function 259
role_name system function 260
round mathematical function 261
row_count system function 263
rtrim string function 264
security 75
set_appcontext security function 266
show_role system function 268
show_sec_services security function 269
sign mathematical function 270
sin mathematical function 271
sortkey 273
sortkey system function 272
soundex string function 277
space string function 278
sqrt mathematical function 281
square mathematical function 280
stddev statistical aggregate function. See stddev_samp.

stddev_pop statistical aggregate function 285
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stddev_samp</code></td>
<td>Statistical aggregate function. See <code>stddev_samp</code>.</td>
</tr>
<tr>
<td><code>stddev</code></td>
<td>Statistical aggregate function. See <code>stddev_samp</code>.</td>
</tr>
<tr>
<td><code>stddevp</code></td>
<td>Statistical aggregate function. See <code>stddev_pop</code>.</td>
</tr>
<tr>
<td><code>str</code></td>
<td>String function 289</td>
</tr>
<tr>
<td><code>str_replace</code></td>
<td>String function 291</td>
</tr>
<tr>
<td><code>stuff</code></td>
<td>String function 295</td>
</tr>
<tr>
<td><code>substring</code></td>
<td>String function 297</td>
</tr>
<tr>
<td><code>sum</code></td>
<td>Aggregate function 299</td>
</tr>
<tr>
<td><code>suser_id</code></td>
<td>System function 301</td>
</tr>
<tr>
<td><code>suser_name</code></td>
<td>System function 302</td>
</tr>
<tr>
<td><code>syb_quit</code></td>
<td>System function 303</td>
</tr>
<tr>
<td><code>syb_sendmsg</code></td>
<td>304</td>
</tr>
<tr>
<td><code>tan</code></td>
<td>Mathematical function 306</td>
</tr>
<tr>
<td><code>tempdb_id</code></td>
<td>System function 307</td>
</tr>
<tr>
<td><code>text</code></td>
<td>78</td>
</tr>
<tr>
<td><code>textptr</code></td>
<td>Text and image function 308</td>
</tr>
<tr>
<td><code>textvalid</code></td>
<td>Text and image function 309</td>
</tr>
<tr>
<td><code>to_unichar</code></td>
<td>String function 310</td>
</tr>
<tr>
<td><code>tran_dumptable_status</code></td>
<td>String function 311</td>
</tr>
<tr>
<td><code>tsequal</code></td>
<td>System function 313</td>
</tr>
<tr>
<td><code>uhighsurr</code></td>
<td>String function 315</td>
</tr>
<tr>
<td><code>ulowsurr</code></td>
<td>String function 316</td>
</tr>
<tr>
<td><code>upper</code></td>
<td>String function 317</td>
</tr>
<tr>
<td><code>uscalar</code></td>
<td>String function 318</td>
</tr>
<tr>
<td><code>used_pages</code></td>
<td>System function 319</td>
</tr>
<tr>
<td><code>user</code></td>
<td>System function 321</td>
</tr>
<tr>
<td><code>user_id</code></td>
<td>System function 322</td>
</tr>
<tr>
<td><code>user_name</code></td>
<td>System function 323</td>
</tr>
<tr>
<td><code>valid_name</code></td>
<td>System function 324</td>
</tr>
<tr>
<td><code>valid_user</code></td>
<td>System function 325</td>
</tr>
<tr>
<td><code>var</code></td>
<td>Statistical aggregate function. See <code>var_samp</code>.</td>
</tr>
<tr>
<td><code>var_pop</code></td>
<td>Statistical aggregate function 327</td>
</tr>
<tr>
<td><code>var_samp</code></td>
<td>Statistical aggregate function 329</td>
</tr>
<tr>
<td><code>variance</code></td>
<td>Statistical aggregate function. See <code>var_samp</code>.</td>
</tr>
<tr>
<td><code>varp</code></td>
<td>Statistical aggregate function. See <code>var_pop</code>.</td>
</tr>
<tr>
<td><code>year</code></td>
<td>Date function 350</td>
</tr>
<tr>
<td><code>getutcdate</code></td>
<td>To obtain the GMT 174</td>
</tr>
<tr>
<td><code>get_appcontext</code></td>
<td>Security function 172</td>
</tr>
<tr>
<td><code>getdate</code></td>
<td>Date function 173</td>
</tr>
<tr>
<td><code>GB Pinyin</code></td>
<td>116, 275</td>
</tr>
</tbody>
</table>

Index
<table>
<thead>
<tr>
<th>Variable</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>@@mempool_addr</code></td>
<td>354</td>
</tr>
<tr>
<td><code>@@min_poolsize</code></td>
<td>355</td>
</tr>
<tr>
<td><code>@@mingroupid</code></td>
<td>355</td>
</tr>
<tr>
<td><code>@@minpid</code></td>
<td>355</td>
</tr>
<tr>
<td><code>@@minuid</code></td>
<td>355</td>
</tr>
<tr>
<td><code>@@minuser</code></td>
<td>355</td>
</tr>
<tr>
<td><code>@@monitors_active</code></td>
<td>355</td>
</tr>
<tr>
<td><code>@@ncharsize</code></td>
<td>355</td>
</tr>
<tr>
<td><code>@@nestlevel</code></td>
<td>355</td>
</tr>
<tr>
<td><code>@@nodeid</code></td>
<td>355</td>
</tr>
<tr>
<td><code>@@optgoal</code></td>
<td>355</td>
</tr>
<tr>
<td><code>@@options</code></td>
<td>355</td>
</tr>
<tr>
<td><code>@@opttimeout</code></td>
<td>355</td>
</tr>
<tr>
<td><code>@@pack_received</code></td>
<td>355</td>
</tr>
<tr>
<td><code>@@pack_sent</code></td>
<td>355</td>
</tr>
<tr>
<td><code>@@packet_errors</code></td>
<td>355</td>
</tr>
<tr>
<td><code>@@pagesize</code></td>
<td>355</td>
</tr>
<tr>
<td><code>@@parallel_degree</code></td>
<td>355</td>
</tr>
<tr>
<td><code>@@probesuid</code></td>
<td>355</td>
</tr>
<tr>
<td><code>@@procid</code></td>
<td>355</td>
</tr>
<tr>
<td><code>@@recovery_state</code></td>
<td>355</td>
</tr>
<tr>
<td><code>@@repartition_degree</code></td>
<td>355</td>
</tr>
<tr>
<td><code>@@resource_granularity</code></td>
<td>356</td>
</tr>
<tr>
<td><code>@@rowcount</code></td>
<td>356</td>
</tr>
<tr>
<td><code>@@scan_parallel_degree</code></td>
<td>356</td>
</tr>
<tr>
<td><code>@@servername</code></td>
<td>356</td>
</tr>
<tr>
<td><code>@@setrowcount</code></td>
<td>356</td>
</tr>
<tr>
<td><code>@@shmem_flags</code></td>
<td>356</td>
</tr>
<tr>
<td><code>@@spid</code></td>
<td>356</td>
</tr>
<tr>
<td><code>@@sqlstatus</code></td>
<td>356</td>
</tr>
<tr>
<td><code>@@ssl_ciphersuite</code></td>
<td>356</td>
</tr>
<tr>
<td><code>@@stringsize</code></td>
<td>356</td>
</tr>
<tr>
<td><code>@@tempdbid</code></td>
<td>356</td>
</tr>
<tr>
<td><code>@@txctoid</code></td>
<td>356</td>
</tr>
<tr>
<td><code>@@txdattphid</code></td>
<td>357</td>
</tr>
<tr>
<td><code>@@txdbufid</code></td>
<td>357</td>
</tr>
<tr>
<td><code>@@txtobjid</code></td>
<td>357</td>
</tr>
<tr>
<td><code>@@txtphinid</code></td>
<td>357</td>
</tr>
<tr>
<td><code>@@txptr</code></td>
<td>357</td>
</tr>
<tr>
<td><code>@@txptr_parameters</code></td>
<td>357</td>
</tr>
<tr>
<td><code>@@txtsize</code></td>
<td>357</td>
</tr>
<tr>
<td><code>@@txts</code></td>
<td>357</td>
</tr>
<tr>
<td><code>@@thresh_hysteresis</code></td>
<td>357</td>
</tr>
<tr>
<td><code>@@timeticks</code></td>
<td>357</td>
</tr>
<tr>
<td><code>@@total_errors</code></td>
<td>357</td>
</tr>
<tr>
<td><code>@@total_read</code></td>
<td>357</td>
</tr>
<tr>
<td><code>@@total_write</code></td>
<td>357</td>
</tr>
<tr>
<td><code>@@tranched</code></td>
<td>357</td>
</tr>
<tr>
<td><code>@@trancount</code></td>
<td>357</td>
</tr>
<tr>
<td><code>@@transactional_rpc</code></td>
<td>357</td>
</tr>
<tr>
<td><code>@@transstate</code></td>
<td>357</td>
</tr>
<tr>
<td><code>@@unicharsize</code></td>
<td>357</td>
</tr>
<tr>
<td><code>@@version</code></td>
<td>357</td>
</tr>
<tr>
<td><code>@@version_as_integer</code></td>
<td>357</td>
</tr>
<tr>
<td><code>@@version_number</code></td>
<td>357</td>
</tr>
<tr>
<td><code>@@datefirst</code></td>
<td>353</td>
</tr>
<tr>
<td><code>@@hacmpservername</code></td>
<td>353</td>
</tr>
<tr>
<td><code>@@haconnection</code></td>
<td>353</td>
</tr>
<tr>
<td><code>has_role</code></td>
<td>175</td>
</tr>
<tr>
<td><code>hash</code></td>
<td>177</td>
</tr>
<tr>
<td><code>having</code></td>
<td>54</td>
</tr>
<tr>
<td><code>hextobigint</code></td>
<td>181</td>
</tr>
<tr>
<td><code>hextoint</code></td>
<td>182</td>
</tr>
<tr>
<td><code>hh</code></td>
<td>74, 150</td>
</tr>
<tr>
<td><code>identifiers</code></td>
<td>369–377</td>
</tr>
<tr>
<td><code>identifiers</code>, case sensitivity and</td>
<td>372</td>
</tr>
<tr>
<td><code>hexdecimal numbers, converting</code></td>
<td>71</td>
</tr>
<tr>
<td><code>historic dates, pre-1753</code></td>
<td>73, 143</td>
</tr>
<tr>
<td><code>host computer name</code></td>
<td>184</td>
</tr>
<tr>
<td><code>host process ID, client process</code></td>
<td>183</td>
</tr>
<tr>
<td><code>host_id</code></td>
<td>183</td>
</tr>
<tr>
<td><code>host_name</code></td>
<td>184</td>
</tr>
<tr>
<td><code>hour</code></td>
<td>74, 150</td>
</tr>
<tr>
<td><code>I</code></td>
<td>369</td>
</tr>
</tbody>
</table>
Index

renaming 376
short 371
system functions and 324
identities
  sa_role and Database Owner 322
  server user (user_id) 302
  user (user_id) 322
  @@identity global variable 353
identity_burn_max function 186
@@idle global variable 354
IDs, server role and role_id 259
IDs, user
database (db_id) 158
  server user 302
  user_id function for 301
image datatype 36–43
  initializing 38
  null values in 39
  prohibited actions on 41
image functions 78
implicit conversion of datatypes 9, 368
in keyword in expressions 365
index_col system function 187
index_colorder system function 188
indexes
  See also clustered indexes; database objects;
    nonclustered indexes
  sysindexes table 39
initializing text or image columns 40
inserting
  automatic leading zero 34
  spaces in text strings 278
instance_name function 198
int datatype 13
  aggregate functions and 88, 300
  integer data in SQL 359
  integer datatypes, converting to 71
  integer remainder. See Modulo operator (%) 
  internal datatypes of null columns 10
  See also datatypes
internal structures, pages used for 250
inttohex datatype conversion function 190
@@invaliduserid global variable 353
@@isbusy global variable 354
is not null keyword in expressions 365
is_quiesced function 193–194
is_sec_service_on security function 195
isnull system function 196
isnumeric function 197
ISO 8859-5 Cyrillic dictionary 116, 276
ISO 8859-5 Russian dictionary 116, 276
ISO 8859-9 Turkish dictionary 116, 276
iso_1 character set 377
@@isolation global variable 354
isql utility command
  See also Utility Guide manual
  approximate numeric datatypes and 17
J
Japanese character sets and object identifiers 377
  joins
    count or count(*) with 127, 128
    null values and 366
K
@@kernel_addr global variable 354
@@kernel_size global variable 354
keywords 385–388
  Transact-SQL 371, 385–386
L
@@langid global variable 354
@@language global variable 354
languages, alternate
  effect on date parts 153
  weekday order and 153
last-chance threshold and lct_admin function 202
last-chance thresholds 203
@@lastlogindate global variable 354
latin-1 English, French, German
  dictionary 116, 275
  no accent 116, 276
latin-1 Spanish
  no accent 116, 276
  no case 116, 276
lc_id function 199
lc_name function 200
lct_admin system function 201, 203
leading blanks, removal with ltrim function 212
leading zeros, automatic insertion of 34
left system function 204
len string function 205
length
See also size
of expressions in bytes 140
identifiers 369
of columns 111
less than. See comparison operators
license_enabled system function 206
like keyword
searching for dates with 25
wildcard characters used with 379
linkage, page. See pages, data
list_appcontext security function 207
listing datatypes with types 7
lists
functions 48
literal character specification
like match string 381
quotes (""") 369
literal values
datatypes of 6
null 367
@@lock_timeout global variable 354
lockscheme system function 208
log mathematical function 208, 209
log10 mathematical function 210
logarithm, base 10 210
logical expressions 359
syntax 360
truth tables for 367
when...then 96, 109, 221
log10 mathematical function 210
longsysname datatype 35
lower and higher datatypes. See precedence.
lower string function 211
lowercase letters, sort order and 372
See also case sensitivity
ltrim string function 212
macintosh character set 377
matching
See also Pattern matching
name and table name 375
mathematical functions 74
abs 80
acos 81
asin 84
atan 85
atan2 86
ceiling 102
cos 124
cot 125
degrees 162
exp 169
floor 170
log 209
log10 210
pi 238
power 239
radians 243
rand 244, 245
round 261
sign 270
sin 271
tan 306
max aggregate function 213
@@max_connections global variable 354
@@max_precision global variable 354
@@maxcharlen global variable 354
@@maxgroupid global variable 354
@@maxpagesize global variable 354
@@maxspid global variable 354
@@maxsuid global variable 354
@@maxuserid global variable 354
@@mempool_addr global variable 354
messages and mathematical functions 75
mi. See minute date part
milliseconds date part 74
midnights, number of 146
millisecond date part 74, 150
millisecond values, datediff results in 146
min aggregate function 215
Index

@@min_poolsize global variable 355
@@mingroupid global variable 355
@@minspid global variable 355
@@minsuid global variable 355

minus sign (-)
  in integer data 13
  subtraction operator 361
@@minuserid global variable 355

minute date part 74, 150
mixed datatypes, arithmetic operations on 361
mm. See month date part
mm. See month date part.
model database, user-defined datatypes in 45
modulo operator (%) 361
money
  default comma placement 18
  symbols 371
money datatype 19
  arithmetic operations and 18
@@monitors_active global variable 355
month date function 216
month date part 73, 150
month values and date part abbreviation 73, 150
ms. See 74
multibyte character sets
  converting 67
  identifier names 376
nchar datatype for 27
  wildcard characters and 381
multiplication operator (*) 361
mut_excl_roles system function 217
mutual exclusivity of roles and mut_excl_roles 217

omitted elements of (..) 375
qualifying database objects 374, 376
user_name function 302
user_name function 323
weekday numbers and 153

naming
  conventions 369–377
  database objects 369–377
  identifiers 369–377
  user-defined datatypes 45
  national character. See nchar datatype
  natural logarithm 208, 209
nchar datatype 28–29
@@ncharsize global variable 355
negative sign (-) in money values 19
nesting
  aggregate functions 55
  string functions 76
@@nestlevel global variable 355
newid system function 218
next_identity system function 220
@@nodeid global variable 355
“none”, using “NULL” or 367
not keyword in expressions 365
not like keyword 378
not null values
  spaces in 31
not null values in spaces 31
null keyword in expressions 365
null string in character columns 296, 367
null values
  column datatype conversion for 31
  default parameters as 366
  in expressions 366
  text and image columns 39
null values in a where clause 366
nullif expressions 221–222
nullif keyword 221
number (quantity of)
  first-of-the-months 146
  midnights 146
  rows in count(*) 126, 128
  Sundays 146
number of characters and date interpretation 26
numbers
  asterisks (***) for overlength 289

N
“N/A”, using “NULL” or 367
names
  See also identifiers
  checking with valid_name 376
date parts 73, 150
db_name function 160
  finding similar-sounding 277
  host computer 184
index_col and index 187
object_name function 228

Adaptive Server Enterprise
converting strings of 32
database ID 158
object ID 227
odd or even binary 34
random float 244, 245
weekday names and 153
numeric data and row aggregates 58
numeric datatype 14
numeric expressions 359
round function for 261
nvarchar datatype 28–29
spaces in 28

O
object names, database
See also identifiers
user-defined datatype names as 45
object_id system function 227
object_name system function 228
object_owner_id=default para font> function 229
objects. See database objects; databases
operators
arithmetic 361
bitwise 362–363
comparison 364
precedence 361
@@optgoal global variable 355
@@options global variable 355
@@opttimeout global variable 355
or keyword in expressions 367
order
See also indexes; precedence; sort order
of execution of operators in expressions 361
of date parts 23
reversing character expression 254
weekday numeric 153
order by clause 273
other users, qualifying objects owned by 376
overflow errors in DB-Library 88, 300
ownership of objects being referenced 376

P
@@pack_received global variable 355
@@pack_sent global variable 355
@@packet_errors global variable 355
padding, data
blanks and 28
underscores in temporary table names 372
with zeros 33
pages, data
chain of 37
used for internal structures 250
pagesize system function 230
@parallel_degree global variable 355
parentheses ()
See also Symbols section of this index
in an expression 368
in SQL statements xx
partition_id function 232
partition_name function 233
partition_object_id function 234
patindex string function 235
text/image function 43
pattern matching 377
See also String functions; wildcard characters
charindex string function 108
difference string function 168
patindex string function 236
percent sign (%)
modulo operator 361
wildcard character 379
period (.)
preceding milliseconds 151
separator for qualifier names 374
pi mathematical function 238
platform-independent conversion
hexadecimal strings to integer values 181, 182
integer values to hexadecimal strings 190
plus (+)
arithmetic operator 361
in integer data 13
null values and 364
string concatenation operator 363
pointers
null for uninitialized text or image column 308
text and image page 308
Index

**text or image column** 38
pound sterling sign (£) 38
in identifiers 371
in money datatypes 19
**power** mathematical function 239
precedence 239
of lower and higher datatypes 368
of operators in expressions 361
preceding blanks. See blanks; spaces, character
precision, datatype 239
approximate numeric types 17
exact numeric types 14
money types 18
@@probesuid global variable 355
**proc_role** system function 240
@@procid global variable 355
**pssinfo** function 242
punctuation, characters allowed in identifiers 371

**Q**
qq. See **quarter** date part
qualifier names 374, 376
**quarter** date part 73, 150
quotation marks (" ")
comparison operators and 364
for empty strings 367, 369
enclosing constant values 76
in expressions 369
literal specification of 369

**R**
radians mathematical function 243
radians, conversion to degrees 162
**rand** mathematical function 244, 245
**rand2**, mathematical function 245
range
See also numbers; size
of date part values 73, 150
datediff results 146
errors in mathematical functions 75
money values allowed 18
of recognized dates 22
wildcard character specification of 380, 381
range queries
and end keyword 365
between start keyword 365
**readtext** command and **text** data initialization
requirement 40
**real** datatype 17
@@recovery_state global variable 355
reference information
datatypes 1
reserved words 385
Transact-SQL functions 47
relational expressions 360
See also comparison operators
removing application contexts 257
@@repartition_degree global variable 355
**replicate** string function 246
**reserve** option, **lct_admin** function 201
**reserve_identity** function 247
reserved words 385–388
See also keywords
database object identifiers and 369, 371
SQL92 386
Transact-SQL 385–386
**reserved_pages** system function 250
@@resource_granularity global variable 356
results of row aggregate operations 57
retrieving similar-sounding words or names 277
reverse string function 254
right string function 255, 256
right-justification of **str** function 290
rm_appcontext security function 257
role hierarchies and **role_contain** 258
role_contain system function 258
role_id system function 259
role_name system function 260
roles
checking with **has_role** 175
checking with **proc_role** 240
showing system with **show_role** 268
roles, user-defined and mutual exclusivity 217
round mathematical function 261
rounding 261
approximate numeric datatypes 17
datetime values 20, 69
money values 18, 68
str string function and 289
row aggregates 57
compute and 57
difference from aggregate functions 58
row_count system function 263
@@rowcount global variable 356
rows, table
detail and summary results 57
row aggregates and 57
rtrim string function 264
rules. See database objects.

S
scalar aggregates and nesting vector aggregates within 55
scale, datatype 14
decimal 9
IDENTITY columns 14
loss during datatype conversion 11
numeric 9
@@scan_parallel_degree global variable 356
scrollable cursor
@@rowcount 353
sdc_intempdbconfig function 265
search conditions and datetime data 25
second date part 74, 150
seconds, datediff results in 146
security functions 75
get_appcontext 172
is_sec_service_on 195
list_appcontex 207
rm_appcontex 257
set_appcontext 266
show_sec_services 269
seed values and rand function 244
select command 273
aggregates and 54
for browse 313
restrictions in standard SQL 55
in Transact-SQL compared to standard SQL 55
select into command not allowed with compute 60
server user name and ID
suser_id function 301
suser_name function for 302
@@servername global variable 356
set_appcontex security function 266
@@setrowcount global variable 356
setting application context 266
shift-JIS binary order 117, 276
@@shmem_flags global variable 356
short identifiers 371
show_role system function 268
show_sec_services security function 269
sign mathematical function 270
similar-sounding words. See soundex string function
sin mathematical function 271
single quotes. See quotation marks
single-byte character sets, char datatype for 27
size
See also length; number (quantity of); range; size
limit; space allocation
column 111
floor mathematical function 171
identifiers (length) 370
image datatype 36
of pi 238
text datatype 36
size limit
approximate numeric datatypes 17
binary datatype 33
char columns 28
datatypes 2
double precision datatype 17
exact numeric datatypes 13
fixed-length columns 28
float datatype 17
image datatype 33
integer value smallest or largest 171
money datatypes 19
nchar columns 29
nvarchar columns 29
real datatype 17
varbinary datatype 33
varchar columns 28
slash (/) division operator 361
smalldatetime datatype 22
date functions and 151
smallint datatype 13
smallmoney datatype 19
sort order
Index

character collation behavior 272, 273
comparison operators and 364
sortkey function 273
sortkey system function 272
soundex string function 277
sp_bindefault system procedure and user-defined datatypes 45
sp_bindrule system procedure and user-defined datatypes 45
sp_help system procedure 45
space string function 278
spaces, character
See also blanks
in character datatypes 28–31
empty strings (‘ ’) or (‘ ’) as 367, 369
inserted in text strings 278
like datetime values and 26
not allowed in identifiers 371
speed (Server)
binary and varbinary datatype access 32
@@spid global variable 356
SQL (used with Sybase databases). See Transact-SQL
SQL standards
aggregate functions and 55
concatenation and 364
SQLSTATE codes 389–395
exceptions 390–395
@@sqlstatus global variable 356
sqrt mathematical function 281
square brackets [ ]
caret wildcard character [*] and 379, 381
in SQL statements xx
wildcard specifier 379
square mathematical function 280
square root mathematical function 281
ss. See second date part
@@ssl_ciphersuite global variable 356
statistical aggregate functions
stddev. See stddev_samp.
stddev_pop 285
stddev_samp 287
stdev. See stddev_samp.
stddevp. See stddev_pop.
var. See var_samp.
var_pop 327
var_samp 329
variance. See var_samp.
varp. See var_pop.
stddev statistical aggregate function. See stddev_samp.
stddev_pop statistical aggregate function 285
stddev_samp statistical aggregate function 287
stdev statistical aggregate function. See stddev_samp.
stddevp statistical aggregate function. See stddev_pop.
storage management for text and image data 39
str string function 289
str_replace string function 291
string functions 76–77
See also text datatype
ascii 82
char 104
char_length 106
charindex 108
difference 168
len 205
lower 211
ltrim 212
patindex 235
replicate 246
reverse 254
right 255
rtrim 264
soundex 277
space 278
str 289
str_replace 291
stuff 295
substring 297
to_unichar 310
tran_dumptable_status 311
uhighsurr 315
ulowsurr 316
upper 317
uscalar 318
strings, concatenating 363
@@stringsize global variable 356
stuff string function 295, 296
style values, date representation 119
subqueries
any keyword and 365
in expressions 365
substring string function 297
subtraction operator (-) 361
sum aggregate function 299
sundays, number value 146
suser_id system function 301
suser_name system function 302
syb_quit system function 303
syb_sendmsg function 304
symbols
See also wildcard characters; Symbols section of this index
arithmetic operator 361
comparison operator 364
in identifier names 371
matching character strings 379
money 371
in SQL statements xix, xx
wildcards 379
synonyms and chars and characters, patindex 235
synonyms for datatypes 2
synonyms, chars and characters, patindex 230
syntax conventions, Transact-SQL xix
syscolumns table 35
sysindexes table and name column in 39
sysname datatype 35
sys srvroles table and role_id system function 259
system datatypes. See datatypes
system functions 77
col_length 111
col_name 112
compare 113
curunreservedpgs 134
data_pages 136–137
datachange 138–139
datalength 140
db_id 158, 160
derived_stat 163
has_role system function 175
hash system function 177
host_id 183
host_name 184
index_col 187
index_colororder 188
isnull 196
lct_admin 201
left 204
license_enabled 206
lockscheme 208
mut_excl_roles 217
newid system function 218
next_identity 220
object_id 227
object_name 228
pagesize 230
proc_role system function 240
reserved_pages 250
role_contain 258
role_id 259
role_name 260
row_count 263
show_role 268
sortkey 272
suser_id 301
suser_name 302
syb_quit 303
tempdb_id 307
tequal 313
used_pages 319
user 321
user_id 322
user_name 323
valid_name 324
valid_user 325
system roles and show_role and 268
system tables and sysname datatype 35
T

table pages
See also pages, data tables
  identifying 374
  names as qualifiers 374
  worktables 54
tan mathematical function 306
tangents, mathematical functions for 306
tempdb database, user-defined datatypes in 45
@@tempdbid global variable 356
tempdb_id system function 307
tempdbs and tempdb_id system function 307
temporary tables, naming 372
  number of bytes 372
  padding 372
Index

sysobjects 372
text and image functions
  textptr 308
textvalid 309

text datatype 36–43
  convert command 42
  converting 68
  initializing with null values 38
  null values 39
  prohibited actions on 41

text datatype and ascii string function 82

text functions 78

text page pointer 111
text pointer values 308
@@textcolid global variable 41, 356
@@textdataptnid global variable 357
@@textdbid global variable 41, 357
@@textobjid global variable 41, 357
@@textptnid global variable 357
textptr function 308
@@textptr global variable 41, 357

textptr text and image function 308
@@textptr_parameters global variable 357
@@textsize global variable 41, 357
@@txsts global variable 41, 357

textvalid text and image function 309
Thai dictionary 116, 275

then keyword. See when...then conditions
@@thresh_hysteresis global variable 357

time values
datatype 20–26
timestamp datatype 19–20
  automatic update of 19
  browse mode and 19, 313
  comparison using tsequal function 313
@@timesticks global variable 357

tinyint datatype 13
to_unichar string function 310
@@total_errors global variable 357
@@total_read global variable 357
@@total_write global variable 357

trailing blanks. See blanks

tran_dumptable_status string function 311
@@tranchained global variable 357
@@trancount global variable 357
@@transactional_rpc global variable 357

Transact-SQL
  aggregate functions in 55
  reserved words 385–386

Transact-SQL extensions 11

translation of integer arguments into binary numbers 362
@@transstate global variable 357

triggers See database objects; stored procedures.

trigonometric functions 74, 74–306

true/false data, bit columns for 35

truncation
  arithaborted numeric_truncation 10
  binary datatypes 32
  character string 28
datediff results 146
  str conversion and 290
temporary table names 372

truth tables for logical expressions 367

tsequal system function 313
twenty-first century numbers 22

U

UDP messaging 304

uhighsurr string function 315

ulowsurr string function 316

underscore (_) character string wildcard 379, 380
  object identifier prefix 324, 370
  in temporary table names 372
@@unicharsize global variable 357

unique names as identifiers 373

unitext datatype 36–43

unsigned bigint datatype 13

unsigned int datatype 13

unsigned smallint datatype 13

updating
  See also changing 19
  in browse mode 313
  prevention during browse mode 313

upper string function 317, 318

uppercase letter preference 372
  See also case sensitivity; order by clause
  us 74
us. See 74
us_english language, weekdays setting 153
uscalar string function 318
used_pages system function 319
User Datagram Protocol messaging 304
user IDs
 user_id function for 322
 valid_user function 325
 user names 323
 user names, finding 302, 323
 user objects. See database objects
 user system function 321
 user_id system function 322
 user_name system function 323
 user-created objects. See database objects
 user-defined datatypes 11
  See also datatypes
  creating 45
  dropping 45
  longsysname as 35
  sysname as 35
 user-defined roles and mutual exclusivity 217
using bytes option, patindex string function 230, 235, 236

V
valid_name system function 324
  using after changing character sets 376
valid_user system function 325
var statistical aggregate function. See var_samp.
var_pop statistical aggregate function 327
var_samp statistical aggregate function 329
varbinary datatype 32–34, 273
varchar datatype 28–29
datetime values conversion to 26
  in expressions 368
  spaces in 28
variable-length character. See varchar datatype
variance statistical aggregate function. See var_samp.
varp statistical aggregate function. See var_pop.
vector aggregates 55
  nesting inside scalar aggregates 55
@@version global variable 357
@@version_number global variable 357

Y
year date function 350
year date part 73, 150
yen sign ($) 73
  in identifiers 371
  in money datatypes 19
yes/no data, bit columns for 35
yy. See year date part

Z
zero x (0x) 71
zeros, trailing, in binary datatypes 33–34
Index