## Contents

**About This Book** ........................................................................................................................... xi

### CHAPTER 1 System and User-Defined Datatypes ................................. 1

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Datatype categories</td>
<td>2</td>
</tr>
<tr>
<td>Range and storage size</td>
<td>2</td>
</tr>
<tr>
<td>Declaring the datatype of a column, variable, or parameter</td>
<td>4</td>
</tr>
<tr>
<td>Declaring the datatype for a column in a table</td>
<td>4</td>
</tr>
<tr>
<td>Declaring the datatype for a local variable in a batch or procedure</td>
<td>5</td>
</tr>
<tr>
<td>Declaring the datatype for a parameter in a stored procedure</td>
<td>5</td>
</tr>
<tr>
<td>Determining the datatype of a literal</td>
<td>5</td>
</tr>
<tr>
<td>Datatype of mixed-mode expressions</td>
<td>6</td>
</tr>
<tr>
<td>Determining the datatype hierarchy</td>
<td>7</td>
</tr>
<tr>
<td>Determining precision and scale</td>
<td>8</td>
</tr>
<tr>
<td>Converting one datatype to another</td>
<td>9</td>
</tr>
<tr>
<td>Automatic conversion of fixed-length NULL columns</td>
<td>9</td>
</tr>
<tr>
<td>Handling overflow and truncation errors</td>
<td>10</td>
</tr>
<tr>
<td>Standards and compliance</td>
<td>11</td>
</tr>
<tr>
<td>Exact numeric datatypes</td>
<td>11</td>
</tr>
<tr>
<td>Function</td>
<td>11</td>
</tr>
<tr>
<td>Integer types</td>
<td>12</td>
</tr>
<tr>
<td>Decimal datatypes</td>
<td>13</td>
</tr>
<tr>
<td>Standards and compliance</td>
<td>14</td>
</tr>
<tr>
<td>Approximate numeric datatypes</td>
<td>15</td>
</tr>
<tr>
<td>Function</td>
<td>15</td>
</tr>
<tr>
<td>Understanding approximate numeric datatypes</td>
<td>15</td>
</tr>
<tr>
<td>Range, precision, and storage size</td>
<td>16</td>
</tr>
<tr>
<td>Entering approximate numeric data</td>
<td>16</td>
</tr>
<tr>
<td>Values that may be entered by Open Client clients</td>
<td>17</td>
</tr>
<tr>
<td>Standards</td>
<td>17</td>
</tr>
<tr>
<td>Money datatypes</td>
<td>17</td>
</tr>
<tr>
<td>Function</td>
<td>17</td>
</tr>
<tr>
<td>Accuracy</td>
<td>17</td>
</tr>
<tr>
<td>Range and storage size</td>
<td>17</td>
</tr>
<tr>
<td>Topic</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Entering monetary values</td>
<td>18</td>
</tr>
<tr>
<td>Standards</td>
<td>18</td>
</tr>
<tr>
<td>Timestamp datatype</td>
<td>18</td>
</tr>
<tr>
<td>Function</td>
<td>18</td>
</tr>
<tr>
<td>Creating a timestamp column</td>
<td>19</td>
</tr>
<tr>
<td>Date and time datatypes</td>
<td>19</td>
</tr>
<tr>
<td>Function</td>
<td>20</td>
</tr>
<tr>
<td>Range and storage requirements</td>
<td>20</td>
</tr>
<tr>
<td>Entering date and time data</td>
<td>21</td>
</tr>
<tr>
<td>Standards and compliance</td>
<td>25</td>
</tr>
<tr>
<td>Character datatypes</td>
<td>25</td>
</tr>
<tr>
<td>Function</td>
<td>25</td>
</tr>
<tr>
<td>Length and storage size</td>
<td>26</td>
</tr>
<tr>
<td>Entering character data</td>
<td>28</td>
</tr>
<tr>
<td>Treatment of blanks</td>
<td>29</td>
</tr>
<tr>
<td>Manipulating character data</td>
<td>30</td>
</tr>
<tr>
<td>Standards</td>
<td>30</td>
</tr>
<tr>
<td>Binary datatypes</td>
<td>31</td>
</tr>
<tr>
<td>Function</td>
<td>31</td>
</tr>
<tr>
<td>Valid binary and varbinary entries</td>
<td>31</td>
</tr>
<tr>
<td>Entries of more than the max column size</td>
<td>31</td>
</tr>
<tr>
<td>Treatment of trailing zeroes</td>
<td>32</td>
</tr>
<tr>
<td>Platform dependence</td>
<td>33</td>
</tr>
<tr>
<td>Standards</td>
<td>33</td>
</tr>
<tr>
<td>bit datatype</td>
<td>33</td>
</tr>
<tr>
<td>Function</td>
<td>33</td>
</tr>
<tr>
<td>Entering data into bit columns</td>
<td>34</td>
</tr>
<tr>
<td>Storage size</td>
<td>34</td>
</tr>
<tr>
<td>Restrictions</td>
<td>34</td>
</tr>
<tr>
<td>Standards</td>
<td>34</td>
</tr>
<tr>
<td>sysname datatype</td>
<td>34</td>
</tr>
<tr>
<td>Function</td>
<td>34</td>
</tr>
<tr>
<td>Using the sysname datatype</td>
<td>35</td>
</tr>
<tr>
<td>Standards</td>
<td>35</td>
</tr>
<tr>
<td>text and image datatypes</td>
<td>35</td>
</tr>
<tr>
<td>Function</td>
<td>35</td>
</tr>
<tr>
<td>Data structures used for storing text and image data</td>
<td>36</td>
</tr>
<tr>
<td>Initializing text and image columns</td>
<td>40</td>
</tr>
<tr>
<td>Saving space by allowing NULL</td>
<td>40</td>
</tr>
<tr>
<td>Getting information from sysindexes</td>
<td>41</td>
</tr>
<tr>
<td>Using readtext and writetext</td>
<td>41</td>
</tr>
<tr>
<td>Determining how much space a column uses</td>
<td>42</td>
</tr>
<tr>
<td>Restrictions on text and image columns</td>
<td>42</td>
</tr>
<tr>
<td>Selecting text and image data</td>
<td>42</td>
</tr>
</tbody>
</table>
Converting text and image datatypes ............................................................. 43
Pattern matching in text data ................................................................. 43
Duplicate rows ....................................................................................... 43
Standards ................................................................................................ 44
User-defined datatypes ........................................................................... 44
    Function ............................................................................................ 44
    Creating frequently used datatypes in the model database ........ 44
    Creating a user-defined datatype ...................................................... 44
    Renaming a user-defined datatype .................................................... 45
    Dropping a user-defined datatype .................................................... 45
    Getting help on datatypes ............................................................... 45
Standards and compliance ....................................................................... 45

CHAPTER 2 Transact-SQL Functions ................................................................ 47
    Types of functions ........................................................................... 47
    Aggregate functions ........................................................................ 52
        Aggregates used with group by ................................................ 53
        Aggregate functions and NULL values .................................... 53
        Vector and scalar aggregates ................................................... 53
        Aggregate functions as row aggregates .................................. 55
    Datatype conversion functions ........................................................ 58
        Converting character data to a non-character type ................. 60
        Converting from one character type to another ..................... 61
        Converting numbers to a character type .................................. 61
        Rounding during conversion to and from money types .......... 61
        Converting date/time information ............................................ 62
        Converting between numeric types ........................................... 62
        Arithmetic overflow and divide-by-zero errors ....................... 63
        Conversions between binary and integer types ....................... 64
        Converting between binary and numeric or decimal types ...... 65
        Converting image columns to binary types .............................. 65
        Converting other types to bit ................................................... 65
        Converting NULL value ........................................................... 66
    Date functions .................................................................................. 66
    Date parts ....................................................................................... 66
    Mathematical functions .................................................................. 67
    Security functions ........................................................................... 69
    String functions ............................................................................... 70
        Limits on string functions ......................................................... 71
    System functions ............................................................................ 71
    Text and image functions ............................................................... 73
    abs .................................................................................................. 74
    acos ................................................................................................ 75
    ascii ............................................................................................... 76
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>asin</td>
<td>77</td>
</tr>
<tr>
<td>atan</td>
<td>78</td>
</tr>
<tr>
<td>atn2</td>
<td>79</td>
</tr>
<tr>
<td>avg</td>
<td>80</td>
</tr>
<tr>
<td>ceiling</td>
<td>82</td>
</tr>
<tr>
<td>char</td>
<td>84</td>
</tr>
<tr>
<td>charindex</td>
<td>86</td>
</tr>
<tr>
<td>char_length</td>
<td>87</td>
</tr>
<tr>
<td>col_length</td>
<td>89</td>
</tr>
<tr>
<td>col_name</td>
<td>90</td>
</tr>
<tr>
<td>compare</td>
<td>91</td>
</tr>
<tr>
<td>convert</td>
<td>95</td>
</tr>
<tr>
<td>cos</td>
<td>100</td>
</tr>
<tr>
<td>cot</td>
<td>101</td>
</tr>
<tr>
<td>count</td>
<td>102</td>
</tr>
<tr>
<td>current_date</td>
<td>104</td>
</tr>
<tr>
<td>current_time</td>
<td>105</td>
</tr>
<tr>
<td>curunreservedpgs</td>
<td>106</td>
</tr>
<tr>
<td>data_pgs</td>
<td>108</td>
</tr>
<tr>
<td>datalength</td>
<td>110</td>
</tr>
<tr>
<td>dateadd</td>
<td>111</td>
</tr>
<tr>
<td>datediff</td>
<td>114</td>
</tr>
<tr>
<td>datename</td>
<td>117</td>
</tr>
<tr>
<td>datepart</td>
<td>119</td>
</tr>
<tr>
<td>day</td>
<td>124</td>
</tr>
<tr>
<td>db_id</td>
<td>125</td>
</tr>
<tr>
<td>db_name</td>
<td>126</td>
</tr>
<tr>
<td>degrees</td>
<td>127</td>
</tr>
<tr>
<td>derived_stat</td>
<td>128</td>
</tr>
<tr>
<td>difference</td>
<td>130</td>
</tr>
<tr>
<td>exp</td>
<td>131</td>
</tr>
<tr>
<td>floor</td>
<td>132</td>
</tr>
<tr>
<td>get_appcontext</td>
<td>134</td>
</tr>
<tr>
<td>getdate</td>
<td>136</td>
</tr>
<tr>
<td>hextoint</td>
<td>137</td>
</tr>
<tr>
<td>host_id</td>
<td>138</td>
</tr>
<tr>
<td>host_name</td>
<td>139</td>
</tr>
<tr>
<td>identity_burn_max</td>
<td>140</td>
</tr>
<tr>
<td>index_col</td>
<td>141</td>
</tr>
<tr>
<td>index_colorder</td>
<td>142</td>
</tr>
<tr>
<td>inttohex</td>
<td>143</td>
</tr>
<tr>
<td>isnull</td>
<td>144</td>
</tr>
<tr>
<td>is_sec_service_on</td>
<td>145</td>
</tr>
<tr>
<td>lct_admin</td>
<td>146</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------</td>
</tr>
<tr>
<td>left</td>
<td>149</td>
</tr>
<tr>
<td>len</td>
<td>151</td>
</tr>
<tr>
<td>license_enabled</td>
<td>152</td>
</tr>
<tr>
<td>list_appcontext</td>
<td>153</td>
</tr>
<tr>
<td>lockscheme</td>
<td>154</td>
</tr>
<tr>
<td>log</td>
<td>155</td>
</tr>
<tr>
<td>log10</td>
<td>156</td>
</tr>
<tr>
<td>lower</td>
<td>157</td>
</tr>
<tr>
<td>ltrim</td>
<td>158</td>
</tr>
<tr>
<td>max</td>
<td>159</td>
</tr>
<tr>
<td>min</td>
<td>161</td>
</tr>
<tr>
<td>month</td>
<td>162</td>
</tr>
<tr>
<td>mut_excl_roles</td>
<td>163</td>
</tr>
<tr>
<td>newid</td>
<td>164</td>
</tr>
<tr>
<td>next_identity</td>
<td>166</td>
</tr>
<tr>
<td>object_id</td>
<td>167</td>
</tr>
<tr>
<td>object_name</td>
<td>168</td>
</tr>
<tr>
<td>pagesize</td>
<td>169</td>
</tr>
<tr>
<td>patindex</td>
<td>171</td>
</tr>
<tr>
<td>pi</td>
<td>174</td>
</tr>
<tr>
<td>power</td>
<td>175</td>
</tr>
<tr>
<td>proc_role</td>
<td>176</td>
</tr>
<tr>
<td>ptn_data_pages</td>
<td>177</td>
</tr>
<tr>
<td>radians</td>
<td>178</td>
</tr>
<tr>
<td>rand</td>
<td>179</td>
</tr>
<tr>
<td>replicate</td>
<td>180</td>
</tr>
<tr>
<td>reserved_pages</td>
<td>181</td>
</tr>
<tr>
<td>reverse</td>
<td>182</td>
</tr>
<tr>
<td>right</td>
<td>183</td>
</tr>
<tr>
<td>rm_appcontext</td>
<td>185</td>
</tr>
<tr>
<td>role_contain</td>
<td>186</td>
</tr>
<tr>
<td>role_id</td>
<td>187</td>
</tr>
<tr>
<td>role_name</td>
<td>188</td>
</tr>
<tr>
<td>round</td>
<td>189</td>
</tr>
<tr>
<td>rowcnt</td>
<td>191</td>
</tr>
<tr>
<td>rtrim</td>
<td>193</td>
</tr>
<tr>
<td>set_appcontext</td>
<td>194</td>
</tr>
<tr>
<td>show_role</td>
<td>196</td>
</tr>
<tr>
<td>show_sec_services</td>
<td>197</td>
</tr>
<tr>
<td>sign</td>
<td>198</td>
</tr>
<tr>
<td>sin</td>
<td>199</td>
</tr>
<tr>
<td>sortkey</td>
<td>200</td>
</tr>
<tr>
<td>soundex</td>
<td>205</td>
</tr>
<tr>
<td>space</td>
<td>206</td>
</tr>
</tbody>
</table>
Contents

square ................................................................. 207
sqrt ................................................................. 208
str .............................................................................. 209
str_replace .......................................................... 211
stuff ......................................................................... 213
substring .............................................................. 215
sum ............................................................................. 217
suser_id .............................................................. 219
suser_name .......................................................... 220
syb_quit ............................................................... 221
syb_sendmsg ....................................................... 222
tan ............................................................................... 223
tempdb_id ............................................................ 224
textptr ................................................................. 225
textvalid ............................................................... 226
to_unichar ............................................................. 227
tsequal ................................................................. 228
uhighsurr .............................................................. 230
ulowsurr ............................................................... 231
upper ......................................................................... 232
uscalar ...................................................................... 233
used_pgs ............................................................... 234
user ............................................................................. 236
user_id ................................................................. 237
user_name ............................................................ 238
valid_name .......................................................... 239
valid_user ............................................................. 240
year .......................................................................... 241

CHAPTER 3  Global Variables ................................................. 243
Adaptive Server’s global variables ........................................ 243

CHAPTER 4  Expressions, Identifiers, and Wildcard Characters ....... 249
Expressions .................................................................. 249
Size of expressions ..................................................... 250
Arithmetic and character expressions .......................... 250
Relational and logical expressions .............................. 250
Operator precedence .................................................. 251
Arithmetic operators ................................................... 251
Bitwise operators ........................................................ 252
String concatenation operator .................................... 253
Comparison operators ............................................... 254
Nonstandard operators ............................................... 254
<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using any, all and in ........................................................................................................ 255</td>
</tr>
<tr>
<td>Negating and testing ............................................................................................................. 255</td>
</tr>
<tr>
<td>Ranges ........................................................................................................................................ 255</td>
</tr>
<tr>
<td>Using nulls in expressions ..................................................................................................... 255</td>
</tr>
<tr>
<td>Connecting expressions .......................................................................................................... 257</td>
</tr>
<tr>
<td>Using parentheses in expressions ......................................................................................... 258</td>
</tr>
<tr>
<td>Comparing character expressions ........................................................................................... 258</td>
</tr>
<tr>
<td>Using the empty string ............................................................................................................ 259</td>
</tr>
<tr>
<td>Including quotation marks in character expressions .......................................................... 259</td>
</tr>
<tr>
<td>Using the continuation character ............................................................................................ 259</td>
</tr>
<tr>
<td>Identifiers ............................................................................................................................... 259</td>
</tr>
<tr>
<td>Tables beginning with # (temporary tables) ............................................................................ 260</td>
</tr>
<tr>
<td>Case sensitivity and identifiers ............................................................................................... 260</td>
</tr>
<tr>
<td>Uniqueness of object names ..................................................................................................... 261</td>
</tr>
<tr>
<td>Using delimited identifiers ....................................................................................................... 261</td>
</tr>
<tr>
<td>Identifying tables or columns by their qualified object name ............................................. 262</td>
</tr>
<tr>
<td>Determining whether an identifier is valid .............................................................................. 264</td>
</tr>
<tr>
<td>Renaming database objects ...................................................................................................... 264</td>
</tr>
<tr>
<td>Using multibyte character sets ............................................................................................... 265</td>
</tr>
<tr>
<td>Pattern matching with wildcard characters ............................................................................ 265</td>
</tr>
<tr>
<td>Using not like ........................................................................................................................... 266</td>
</tr>
<tr>
<td>Case and accent insensitivity ................................................................................................... 267</td>
</tr>
<tr>
<td>Using wildcard characters ........................................................................................................ 267</td>
</tr>
<tr>
<td>Using multibyte wildcard characters ....................................................................................... 269</td>
</tr>
<tr>
<td>Using wildcard characters as literal characters ..................................................................... 269</td>
</tr>
<tr>
<td>Using wildcard characters with datetime data ....................................................................... 271</td>
</tr>
<tr>
<td>CHAPTER 5 Reserved Words ..................................................................................................... 273</td>
</tr>
<tr>
<td>Transact-SQL reserved words .................................................................................................. 273</td>
</tr>
<tr>
<td>ANSI SQL reserved words ........................................................................................................ 274</td>
</tr>
<tr>
<td>Potential ANSI SQL reserved words ....................................................................................... 275</td>
</tr>
<tr>
<td>CHAPTER 6 SQLSTATE Codes and Messages ............................................................................. 277</td>
</tr>
<tr>
<td>Warnings ................................................................................................................................. 277</td>
</tr>
<tr>
<td>Exceptions ............................................................................................................................... 278</td>
</tr>
<tr>
<td>Cardinality violations .............................................................................................................. 278</td>
</tr>
<tr>
<td>Data exceptions ......................................................................................................................... 279</td>
</tr>
<tr>
<td>Integrity constraint violations ................................................................................................. 280</td>
</tr>
<tr>
<td>Invalid cursor states ................................................................................................................ 280</td>
</tr>
<tr>
<td>Syntax errors and access rule violations .................................................................................. 281</td>
</tr>
<tr>
<td>Transaction rollbacks ............................................................................................................... 282</td>
</tr>
<tr>
<td>with check option violation .................................................................................................... 282</td>
</tr>
</tbody>
</table>

Reference Manual: Building Blocks ix
Contents

Index ........................................................................................................................................... 285
About This Book

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 sucessfully, you need to 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,” which 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," which provides information about the Transact-SQL and ANSI SQL keywords.

Chapter 6, "SQLSTATE Codes and Messages," which contains information about Adaptive Server’s SQLSTATE status codes and the associated messages.

**Related documents**

The Sybase Adaptive Server Enterprise documentation set consists of the following:

- 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 on the World Wide Web. To check for critical product or document information that was added after the release of the product CD, use the Sybase Technical Library.

- The *Installation Guide* for your platform – describes installation, upgrade, and configuration procedures for all Adaptive Server and related Sybase products.

- *What's New in Adaptive Server Enterprise?* – describes the new features in Adaptive Server version 12.5.1, the system changes added to support those features, and the changes that may affect your existing applications.

- *ASE Replicator User’s Guide* – describes how to use the ASE Replicator feature of Adaptive Server to implement basic replication from a primary server to one or more remote Adaptive Servers.

- *Component Integration Services User's Guide* – explains how to use the Adaptive Server Component Integration Services feature to connect remote Sybase and non-Sybase databases.

- *Configuring Adaptive Server Enterprise* for your platform – provides instructions for performing specific configuration tasks for Adaptive Server.


- *Error Messages and Troubleshooting Guide* – explains how to resolve frequently occurring error messages and describes solutions to system problems frequently encountered by users.

• Glossary – defines technical terms used in the Adaptive Server documentation.


• Java in Adaptive Server Enterprise – describes how to install and use Java classes as data types, functions, and stored procedures in the Adaptive Server database.

• Job Scheduler User’s 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).

• Monitor Client Library Programmer’s Guide – describes how to write Monitor Client Library applications that access Adaptive Server performance data.


• Performance and Tuning Guide – is a series of four books that explains how to tune Adaptive Server for maximum performance:
  • Basics – the basics for understanding and investigating performance questions in Adaptive Server.
  • Locking – describes how the various locking schemas can be used for improving performance in Adaptive Server.
  • Optimizer and Abstract Plans – describes how the optimizer processes queries and how abstract plans can be used to change some of the optimizer plans.
  • Monitoring and Analyzing – explains how statistics are obtained and used for monitoring and optimizing performance.

• 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.

• Reference Manual – is a series of four books that contains the following detailed Transact-SQL® information:
  • Building Blocks – Transact-SQL datatypes, functions, global variables, expressions, identifiers and wildcards, and reserved words.
  • Commands – Transact-SQL commands.
• Procedures – Transact-SQL system procedures, catalog stored procedures, system extended stored procedures, and dbcc stored procedures.

• Tables – Transact-SQL system tables and dbcc tables.

• System Administration Guide – provides in-depth information about administering servers and databases. This manual includes instructions and guidelines for managing physical resources, security, user and system databases, and specifying character conversion, international language, and sort order settings.

• System Tables Diagram – illustrates system tables and their entity relationships in a poster format. Available only in print version.

• Transact-SQL User’s Guide – documents Transact-SQL, Sybase’s enhanced version of the relational database language. This manual serves as a textbook for beginning users of the database management system. This manual also contains descriptions of the pubs2 and pubs3 sample 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 Sybase Failover in a High Availability System – provides instructions for using Sybase’s Failover to configure an Adaptive Server as a companion server in a high availability system.

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


• 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 comprise XML Services.

Other sources of information

Use the Sybase Getting Started CD, the Sybase Technical Library CD and the Technical Library 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 Technical Library CD. It is included with your software. To read or print documents on the Getting Started CD you need Adobe Acrobat Reader (downloadable at no charge from the Adobe Web site, using a link provided on the CD).

• The Technical Library CD contains product manuals and is included with your software. The DynaText reader (included on the Technical Library CD) allows you to access technical information about your product in an easy-to-use format.

Refer to the Technical Library Installation Guide in your documentation package for instructions on installing and starting the Technical Library.

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

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

Technical documentation at the Sybase Web site is updated frequently.

❖ Finding the latest information on product certifications

1 Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.

2 Select Products from the navigation bar on the left.

3 Select a product name from the product list and click Go.

4 Select the Certification Report filter, specify a time frame, and click Go.

5 Click a Certification Report title to display the report.

❖ 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.

1 Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.

2 Click MySybase and create a MySybase profile.
Finding the latest information on EBFs and software maintenance


2. Select EBFs/Maintenance. Enter user name and password information, if prompted (for existing Web accounts) or create a new account (a free service).

3. Select a product.

4. Specify a time frame and click Go.

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, command options, utility names, utility options, and other keywords are in “command” font (Arial, 8 point).</td>
<td>select&lt;br&gt;sp_configure&lt;br&gt;master database</td>
</tr>
<tr>
<td>Database names, datatypes, file names and path names are in “database object” font (Arial, 8 point).</td>
<td>System Administration Guide&lt;br&gt;sql.ini file&lt;br&gt;column_name&lt;br&gt;$SYBASE/ASE directory</td>
</tr>
<tr>
<td>Book names, file names, variables, and path names are in italics.</td>
<td>System Administration Guide&lt;br&gt;sql.ini file&lt;br&gt;column_name&lt;br&gt;$SYBASE/ASE directory</td>
</tr>
<tr>
<td>Variables, or words that stand for values that you fill in, are in “variable” font (Italics).</td>
<td>select column_name from table_name where search_conditions</td>
</tr>
<tr>
<td>Type parentheses as part of the command.</td>
<td>compute row_aggregate (column_name)</td>
</tr>
</tbody>
</table>
Syntax statements (displaying the syntax and all options for a command) appear as follows:

```
sp_dropdevice [device_name]
```

For a command with more options:

```
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:

```
select * from publishers
```

- Examples of output from the computer appear as follows:

```
<table>
<thead>
<tr>
<th>pub_id</th>
<th>pub_name</th>
<th>city</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>0736</td>
<td>New Age Books</td>
<td>Boston</td>
<td>MA</td>
</tr>
</tbody>
</table>
```
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’s 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.

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. Datatypes specify the type, size, and storage format of columns, stored procedure parameters, and local variables.

Topics covered are:

<table>
<thead>
<tr>
<th>Topics</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Datatype categories</td>
<td>2</td>
</tr>
<tr>
<td>Range and storage size</td>
<td>2</td>
</tr>
<tr>
<td>Declaring the datatype of a column, variable, or parameter</td>
<td>4</td>
</tr>
<tr>
<td>Datatype of mixed-mode expressions</td>
<td>6</td>
</tr>
<tr>
<td>Converting one datatype to another</td>
<td>9</td>
</tr>
<tr>
<td>Standards and compliance</td>
<td>11</td>
</tr>
<tr>
<td>Exact numeric datatypes</td>
<td>11</td>
</tr>
<tr>
<td>Approximate numeric datatypes</td>
<td>15</td>
</tr>
<tr>
<td>Money datatypes</td>
<td>17</td>
</tr>
<tr>
<td>Timestamp datatype</td>
<td>18</td>
</tr>
<tr>
<td>Date and time datatypes</td>
<td>19</td>
</tr>
<tr>
<td>Character datatypes</td>
<td>25</td>
</tr>
<tr>
<td>Binary datatypes</td>
<td>31</td>
</tr>
<tr>
<td>bit datatype</td>
<td>33</td>
</tr>
<tr>
<td>sysname datatype</td>
<td>34</td>
</tr>
<tr>
<td>text and image datatypes</td>
<td>35</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 and sysname. Table 1-1 lists the categories of Adaptive Server datatypes. Each category is described in a section of this chapter.

<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 datatype</td>
<td>System tables</td>
</tr>
<tr>
<td>text and image datatypes</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>User-defined datatypes</td>
<td>Defining objects that inherit the rules, default, null type, IDENTITY property, and base datatype</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>
<thead>
<tr>
<th>Datatypes</th>
<th>Synonyms</th>
<th>Range</th>
<th>Bytes of storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact numeric datatypes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tinyint</td>
<td></td>
<td>0 to 255</td>
<td>1</td>
</tr>
<tr>
<td>smallint</td>
<td></td>
<td>-2^15 (-32,768) to 2^15 -1 (32,767)</td>
<td>2</td>
</tr>
</tbody>
</table>
### CHAPTER 1  System and User-Defined Datatypes

<table>
<thead>
<tr>
<th>Datatypes</th>
<th>Synonyms</th>
<th>Range</th>
<th>Bytes of storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>integer</td>
<td>(-2^{31} ) to (2^{31} - 1)</td>
<td>4</td>
</tr>
<tr>
<td>numeric (p, s)</td>
<td></td>
<td>(-10^{38} ) to (10^{38} - 1)</td>
<td>2 to 17</td>
</tr>
<tr>
<td>decimal (p, s)</td>
<td>dec</td>
<td>(-10^{38} ) to (10^{38} - 1)</td>
<td>2 to 17</td>
</tr>
</tbody>
</table>

*Approximate numeric datatypes*

- float (precision)
- double precision
- real

<table>
<thead>
<tr>
<th>Money datatypes</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>smallmoney</td>
<td></td>
<td>(-214,748.3648 ) to (214,748.3647)</td>
<td>4</td>
</tr>
<tr>
<td>money</td>
<td></td>
<td>(-922,337,203,685,477.5808 ) to (922,337,203,685,477.5807)</td>
<td>8</td>
</tr>
</tbody>
</table>

*Date/time datatypes*

- smalldatetime
- datetime
- date
- time

<table>
<thead>
<tr>
<th>Character datatypes</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>char(n)</td>
<td>character</td>
<td>Determined by your server’s logical page size</td>
<td>(n)</td>
</tr>
<tr>
<td>varchar(n)</td>
<td>char[acter] varying</td>
<td>Determined by your server’s logical page size</td>
<td>actual entry length</td>
</tr>
</tbody>
</table>
| unichar             | Unicode character | Determined by your server’s logical page size                        | \(n \times @\text{@\texttt{unicharsize}}\)  
                    |                               | (@\texttt{unicharsize} equals 2)                       |
| univarchar          | Unicode character varying | Determined by your server’s logical page size                        | actual number of characters \(\times @\text{@\texttt{unicharsize}}\)  
                    |                               | \(n \times @\text{@\texttt{ncharsize}}\)                       |
| nchar(n)            | national char[acter] | Determined by your server’s logical page size                        | \(n\)            |
| nvchar(n)           | nchar varying, national char[acter] varying | Determined by your server’s logical page size                        | \(n\)            |

*Binary datatypes*

- binary(n)
- varbinary(n)

*Bit datatype*
Declaring the datatype of a column, variable, or parameter

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.

Declaring the datatype for a column in a table

Use the following syntax to declare the datatype of a new column in a create table or an alter table statement:

```sql
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:

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

<table>
<thead>
<tr>
<th>Datatypes</th>
<th>Synonyms</th>
<th>Range</th>
<th>Bytes of storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit</td>
<td></td>
<td>0 or 1</td>
<td>1 (1 byte holds up to 8 bit columns)</td>
</tr>
</tbody>
</table>

**Text and image datatypes**

- **text**: $2^{31} - 1$ (2,147,483,647) bytes or fewer, 0 until initialized, then a multiple of the logical page size
- **image**: $2^{31} - 1$ (2,147,483,647) bytes or fewer, 0 until initialized, then a multiple of the logical page size
CHAPTER 1  System and User-Defined Datatypes

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

Use the following syntax to declare the datatype for a local variable in a batch or stored procedure:

```
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 [;number]
     [([@]@parameter_name datatype [= default] [output]
         [, @parameter_name datatype [= default]
             [output]]...[])]
     [with recompile]
     as SQL_statements
```

For example:

```
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.
Datatype of mixed-mode expressions

- 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

Prior to Adaptive Server version 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 in 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 “jis” (Japanese) client. For example:

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

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

Datatype 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>
<tr>
<td>decimaln</td>
<td>8</td>
</tr>
<tr>
<td>decimal</td>
<td>9</td>
</tr>
<tr>
<td>moneyn</td>
<td>10</td>
</tr>
<tr>
<td>money</td>
<td>11</td>
</tr>
<tr>
<td>smallmoney</td>
<td>12</td>
</tr>
<tr>
<td>smalldatetime</td>
<td>13</td>
</tr>
<tr>
<td>intn</td>
<td>14</td>
</tr>
<tr>
<td>int</td>
<td>15</td>
</tr>
<tr>
<td>smallint</td>
<td>16</td>
</tr>
<tr>
<td>tinyint</td>
<td>17</td>
</tr>
<tr>
<td>bit</td>
<td>18</td>
</tr>
<tr>
<td>univarchar</td>
<td>19</td>
</tr>
<tr>
<td>unichar</td>
<td>20</td>
</tr>
<tr>
<td>sysname</td>
<td>22</td>
</tr>
<tr>
<td>varchar</td>
<td>22</td>
</tr>
<tr>
<td>nvarchar</td>
<td>22</td>
</tr>
<tr>
<td>char</td>
<td>23</td>
</tr>
<tr>
<td>nchar</td>
<td>23</td>
</tr>
<tr>
<td>timestamp</td>
<td>24</td>
</tr>
<tr>
<td>varbinary</td>
<td>24</td>
</tr>
<tr>
<td>binary</td>
<td>25</td>
</tr>
<tr>
<td>text</td>
<td>26</td>
</tr>
<tr>
<td>image</td>
<td>27</td>
</tr>
<tr>
<td>date</td>
<td>28</td>
</tr>
</tbody>
</table>
Datatype of mixed-mode expressions

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>29</td>
</tr>
<tr>
<td>daten</td>
<td>30</td>
</tr>
<tr>
<td>timen</td>
<td>31</td>
</tr>
<tr>
<td>extended type</td>
<td>99</td>
</tr>
</tbody>
</table>

(35 rows affected)

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.

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 16; royalty is an int, which has a hierarchy of 15. Therefore, the datatype of the result is an int:

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

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 * 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>

Converting one datatype to another

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, and inttohex functions. See “Datatype conversion functions” on page 58 for details about datatype conversions supported by Adaptive Server.

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:
Converting one datatype to another

Table 1-4: Automatic conversion of fixed-length datatypes

<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>nvchar</td>
</tr>
<tr>
<td>binary</td>
<td>varbinary</td>
</tr>
<tr>
<td>datetime</td>
<td>datetimen</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>int, smallint, and tinyint</td>
<td>intn</td>
</tr>
<tr>
<td>decimal</td>
<td>decimain</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.

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.

Note The arithabort and arithignore options were redefined for release 10.0. If you use these options in your applications, examine them to be sure they still produce the desired effects.

Standards and compliance

ANSI SQL – Compliance level: Transact-SQL provides the smallint, int, numeric, decimal, float, double precision, real, char, varchar, date and time ANSI SQL datatypes. The tinyint, binary, varbinary, image, bit, datetime, smalldatetime, money, smallmoney, nchar, nvarchar, unichar, univarchar, sysname, text, timestamp, and user-defined datatypes are Transact-SQL extensions.

Exact numeric datatypes

Function

Use the exact numeric datatypes when it is important to represent a value exactly. Adaptive Server provides exact numeric types for both integers (whole numbers) and numbers with a decimal portion.
Exact numeric datatypes

Integer types

Adaptive Server provides three exact numeric datatypes to store integers: int (or integer), smallint, and tinyint. 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 1-5:

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Stores</th>
<th>Bytes of storage</th>
</tr>
</thead>
<tbody>
<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>
</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 and integer datatypes can be preceded by an optional plus or minus sign. The tinyint datatype can be preceded by an optional plus sign.

Table 1-6 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>2.000</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 1-7 lists some invalid entries for an integer column:
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 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:

\[ \text{datatype} \left[ (\text{precision} [, \text{scale}]) \right] \]

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:

\[ \text{ceiling} \left( \text{precision} / \log_{256} \right) + 1 \]

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

---

**Table 1-7: Invalid integer values**

<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>
**Exact numeric datatypes**

**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 1-8 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 1-9 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, numeric, and decimal ANSI SQL exact numeric datatypes. The tinyint type is a Transact-SQL extension.
Approximate numeric datatypes

Function

Use the approximate numeric types, float, double precision, and real, for numeric data that can tolerate rounding during arithmetic operations. 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 except modulo.

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) can not (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.
Approximate numeric datatypes

If you begin with numbers that are almost correct, and do 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 1-10 shows the range and storage size for each approximate numeric type. Note that `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><code>float[(default precision)]</code></td>
<td>4 for default precision &lt; 16&lt;br&gt;8 for default precision ≥ 16</td>
</tr>
<tr>
<td><code>double precision</code></td>
<td>8</td>
</tr>
<tr>
<td><code>real</code></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.
Values that may be entered by Open Client clients

“NaN” and “Inf” are special values that the floating point number standard uses to represent values that are “not a number” and “infinity,” respectively. Adaptive Server does not usually permit, and does not check for, these values, but Open Client clients can sometimes force these values into tables.

Standards

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

Money datatypes

Function

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 1-11 summarizes the range and storage requirements for money datatypes:

Reference Manual: Building Blocks 17
### Timestamp datatype

**Table 1-11: 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**

ANSI SQL – The `money` and `smallmoney` datatypes are Transact-SQL extensions.

### Timestamp datatype

**Function**

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)
```

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 could be confusing to other users and would 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

Adaptive Server has various ways to identify date and time. Prior to version 12.5.1, only *datetime* and *smalldatetime* were available. As of version 12.5.1, date and time have been added as separate datatypes.

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Date range</th>
<th>Storage size</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>January 1, 0001 to December 31, 9999</td>
<td>4</td>
</tr>
<tr>
<td>time</td>
<td>12:00:00:00 AM to 11:59:59.999 PM</td>
<td>4</td>
</tr>
<tr>
<td>smalldatetime</td>
<td>January 1, 1900 to June 6, 2079</td>
<td>4</td>
</tr>
<tr>
<td>datetime</td>
<td>January 1, 1753 to December 31, 9999</td>
<td>8</td>
</tr>
</tbody>
</table>

Enclose date and time information in single or double quotes. You can enter it in either uppercase or lowercase letters and include spaces between data parts. Adaptive Server recognizes a wide variety of data entry formats; however, Adaptive Server rejects values such as 0 or 00/00/00, which are not recognized as dates.
The default display format for dates is “Apr 15 1987 10:23PM”. You can use the convert function for other styles of date display. You can also do 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.

- **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 upper or lower case.

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

**Function**

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

**Range and storage requirements**

Table 1-12 summarizes the range and storage requirements for the datetime, smalldatetime, date and time datatypes:
Entering date and time data

The datetime and smalldatetime 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. The values must be enclosed in single or double quotes.

- datetime columns hold dates between January 1, 1753 and December 31, 9999. datetime values are accurate to 1/300 of a 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. Storage size is 4 bytes: 2 bytes for the number of days since January 1, 1900 and 2 bytes for the number of minutes since midnight.
- date columns hold dates from January 1, 0001 to December 31, 9999. Storage size is 4 bytes.
- time columns hold time in hours, minutes, seconds and milliseconds. The range 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 upper or lower case.

Entering the date

Dates consist of a month, day, and year and can be entered in a variety of formats for date, datetime 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.
Date and time datatypes

- 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.

Table 1-13 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></td>
<td></td>
<td>“520128”</td>
<td>Jan 28 1952</td>
</tr>
<tr>
<td>8-digit string with no separators</td>
<td>Interpreted as yyyymmd.</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” “4.15.94” “04.15/94”</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>
</tbody>
</table>
CHAPTER 1 System and User-Defined Datatypes

Entering the time

The time component of a datetime, smalldatetime or time 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.
- If you omit the time portion of a datetime or smalldatetime value, Adaptive Server uses the default time of 12:00:00:000AM.

Displaying formats for datetime, smalldatetime, date values

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 1-14 lists some examples of datetime entries and their display values:
### Table 1-14: Examples of datetime and date entries

<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>

### Displaying formats for time value

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

### Table 1-15: Examples of time entries

<table>
<thead>
<tr>
<th>Entry</th>
<th>Value displayed</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;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>

### Finding values that match a pattern

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.

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:

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

You can find the entry using the like operator:

```sql
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, date and time values. Since the standard display formats do not include seconds or milliseconds, you cannot search for seconds or milliseconds with like and a match 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 66.

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

Function

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.
### Character datatypes

- 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 singlebyte 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 just \( n \) bytes of storage.

- Character datatypes can store a maximum of a pagesize worth of data.

- Use the `text` datatype (described in text and image datatypes)—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, 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.

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`.

See Chapter 7, “Configuring Character Sets, Sort Orders, and Languages,” in the *System Administration Guide* for more information about Unicode.

### 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 \texttt{char} and \texttt{varchar} datatypes. For \texttt{unichar}, use \( n \) to specify the number of unicode characters (the amount of storage allocated is 2 bytes per character). For \texttt{nchar} and \texttt{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 \texttt{create table}, \texttt{alter table}, and variables created with \texttt{declare}.

- The default length is 30 bytes for values created with the \texttt{convert} function.

Entries shorter than the assigned length are blank-padded; entries longer than the assigned length are truncated without warning, unless the \texttt{string_rtruncation} option to the \texttt{set} command is set to \texttt{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, \texttt{varchar(n)}, \texttt{univarchar(n)}, and \texttt{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 1-16 summarizes the storage requirements of the different character datatypes:

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

Use the \texttt{char_length} string function and \texttt{datalength} system function to determine column length:

- \texttt{char_length} returns the number of characters in the column, stripping trailing blanks for variable-length datatypes.

- \texttt{datalength} returns the number of bytes, stripping trailing blanks for data stored in variable-length columns.
When a char value is declared to allow NULLS, 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.

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

**Entering Unicode characters**

Optional new syntax added in Adaptive Server 12.5.1 allows you to specify arbitrary Unicode characters. If a character literal is immediately preceded by U& or u& (with no intervening whitespace), the parser recognizes escape sequences within the literal. An escape sequence of the form \xxxx (where xxxx represents 4 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:

```
select * from mytable where unichar_column = U&'\4e94'
```
is equivalent to:

```
select * from mytable where unichar_column = 'Hi.'
```

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 entirely equivalent:

```
select * from mytable where char_column = 'A'
select * 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
insert spaces values ("a", "b", "c", "d",
                   "pads char-not-null only")
insert spaces values ("1   ", "2    ", "3   ",
                   "4    ", "truncates trailing blanks")
insert spaces values ("   e", "   f", "   g",
                   "   h", "leading blanks, no change")
insert spaces values ("w   ", "   x", "   y",
                   "   z", "truncates trailing blanks")
insert spaces values ("", "", "", "",
                   "empty string equals space")
```

```sql
select "[" + cnot + "]",
      "[" + cnull + "]",
      "[" + vnot + "]",
      "[" + vnull + "]",
```

select * from mytable where unichar_column = '    '
Character datatypes

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>explanation from spaces explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>pads char-not-null only</td>
</tr>
<tr>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>truncates trailing blanks</td>
</tr>
<tr>
<td>w</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>leading blanks, no change</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>empty string equals space</td>
</tr>
</tbody>
</table>

(5 rows affected)

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.

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. Strings consisting of numbers can be used for arithmetic after being converted to exact and approximate numeric datatypes with the convert function.

Standards

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.
CHAPTER 1 System and User-Defined Datatypes

Binary datatypes

Function

Use the binary datatypes, binary(n) and varbinary(n), to store raw binary data, such as pictures, in a hexadecimal-like 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 max 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 the section “text and image datatypes.”
**Binary datatypes**

**Treatment of trailing zeroes**

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.

```sql
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)
```

```sql
select * from zeros
bnot             bnull        vnot        vnull
------------     ---------    ----------  ---------
0x1234500000     0x123450     0x123450    0x123450
0x0123000000     0x0123       0x0123      0x0123
```

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:

```sql
insert zeros values (0x0, 0x0,0x0, 0x0)
```

```sql
select * from zeros where bnot = 0x00
bnot             bnull        vnot        vnull
----------     ------      -----       ------------
0x0000000000     0x00        0x00        0x00
```

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 `inttohex` and `hextoint` functions rather than the platform-specific `convert` function. For details, see “Datatype conversion functions”.

Standards

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

*bit* datatype

Function

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.
### sysname datatype

#### Entering data into *bit* 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

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.

#### Restrictions

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

#### Standards

ANSI SQL – Compliance level: Transact-SQL extension.

### sysname datatype

#### Function

*sysname* is a user-defined datatype that is distributed on the Adaptive Server installation tape and used in the system tables. Its definition is:

```sql
varchar(30) "not null"
```
Using the *sysname* datatype

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

Standards

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

*text* and *image* datatypes

**Function**

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

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

You define a text or image column as you would any other column, with a *create table* or *alter table* statement. text and image datatype definitions do not include lengths. They do permit null values. The column definition takes the form:

\[ column\_name \{text | image\} \[null\] \]

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:

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

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

```sql
create table au_pix
(au_id char(11) not null,
 pic image null,
```
text and image datatypes

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

<table>
<thead>
<tr>
<th>Putting additional pages on another device</th>
</tr>
</thead>
<tbody>
<tr>
<td>You can place subsequent text and image data pages on a different logical device with <code>sp_placeobject</code>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Zero padding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image values that have an odd number of hexadecimal digits are padded with a leading zero (an insert of “0xaaabb” becomes “0x0aaabb”).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Effect of partitioning on data storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>You can use the partition option of the <code>alter table</code> command to partition a table that contains text 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 and image columns are stored.</td>
</tr>
</tbody>
</table>

Data structures used for storing text and image data

When you allocate text 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 or image data. This text pointer is known as the first text page (FTP).

The FTP contains two parts:

- The text data page chain, which contains the your 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 FTP is allocated for text or image data, it is never deallocated. If an update to an existing text or image data row results in fewer text pages than are currently allocated for this text or image data, Adaptive Server deallocates the extra text pages. If an update to text or image data sets the value to NULL, all pages except the FTP are deallocated.

Figure 1-1 shows the relationship between the datarow and the text pages
In Figure 1-1, columns \( c_{\text{text}} \) and \( c_{\text{image}} \) are text and image columns containing the pages at the bottom of the picture.

**Format of text data pages**

Each text data page contains user text and image data, and a section known as the text and image pages stats area (TIPSA).

The TIPSA contains information about the text and image data that is contained on the current text page. For instance, in a server configured for multibyte character sets, the TIPSA contains the number of whole characters that are on the current page.

On the FTP, there is an additional area with contains the head of the text node data structure. This area is known as the L0 cache. The text node data structure is described below.

Figure 1-2 describes the format of a FTP:
Text nodes

A text node is a hierarchical tree data structure that maps byte offsets (and character offsets for multibyte servers) to text pages for text data. Text nodes are used for:

- Text-page prefetch
- Indexing to text or image data when starting offsets are specified for readtext

Each entry in the text node points to the text or image data page where a byte offset (or character for multibyte servers) begins. Using this data structure, when given an offset into text/image data, the starting page can be determined, and the text or image data is read starting at that offset. This eliminates the need of having to start at the beginning of the text or image data and discarding all of the data the comes before the offset.

Text nodes take advantage of the fact that text or image data pages are typically allocated with multiple runs of consecutive page numbers. This means there does not need to be a one to one correspondence between the pages allocated to the text or image data, and the number of entries in the text node, which results in reducing the number of pages that are allocated to the text or image data.
Figure 1-3 describes this compression:

**Figure 1-3: How text or image page numbers are allocated**

<table>
<thead>
<tr>
<th>Pages 300 to 310</th>
<th>Pages 330 to 345</th>
<th>Pages 392 to 411</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTP</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In this example, the text or image data is made up of 87 text or image pages, but because there are three separate runs of consecutive page numbers, (300 to 310), (330 to 345), and (392 to 411), only three text node entries are needed, not 87.

The text node is saved with the text or image data. Depending on the size of the text node, extra text or image pages may be required to store the text node. The size of the text node depends on the size of the text or image data, and the amount of ‘compression’ achieved. Although smaller text nodes do not require extra text or image pages, larger text nodes will require them.

The head of the text node, the L0-cache, is stored on the FTP.

Figure 1-4 describes the structure of a text node. L0 cache is the text node, and L1 and L2 are indirect nodes that point to text or image data pages.

**Figure 1-4: Structure of the text node**
Initializing text and image columns

text 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 or image data value. It also creates a pointer in the table to the location of the text or image data.

For example, the following statements create the table testtext 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 testtext
    (title_id varchar(6), blurb text null, pub_id char(4))
insert testtext 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** Remember to surround text values with quotation marks and precede image values with the characters “0x”.

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

Saving space by allowing NULL

To save storage space for empty text 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 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:

```sql
insert testtext
    (title_id, pub_id) values ("BU7832", "1389")
```
CHAPTER 1 System and User-Defined Datatypes

After a text or image row is given a non-null value, it always contains at least one data page. Resetting the value to null does not deallocate its data page.

Getting information from sysindexes

Each table with text 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:

<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 blurbs table in the pubs2 database:

```sql
select name, data_pgs(object_id("blurbs"), ioampg)
from sysindexes
where name = "tblurbs"
```

<table>
<thead>
<tr>
<th>name</th>
<th>data_pgs(object_id(&quot;blurbs&quot;), ioampg)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tblurbs</td>
<td>7</td>
</tr>
</tbody>
</table>

Note The system tables poster shows a one-to-one (1-1) 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.
Using update to replace existing text 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.

Determining how much space a column uses

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

<table>
<thead>
<tr>
<th>sp_spaceused blurs</th>
<th>name</th>
<th>rowtotal</th>
<th>reserved</th>
<th>data</th>
<th>index_size</th>
<th>unused</th>
</tr>
</thead>
<tbody>
<tr>
<td>blurs</td>
<td>blurs</td>
<td>6</td>
<td>32 KB</td>
<td>2 KB</td>
<td>14 KB</td>
<td>16 KB</td>
</tr>
</tbody>
</table>

Restrictions on text and image columns

text and image columns cannot be used:

- 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
- In the if update clause of a trigger

Selecting text and image data

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

Table 1-18: text and image global variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>@@textptr</td>
<td>The text pointer of the last text or image column inserted or updated by a process. Do not confuse this global variable with the textptr function.</td>
</tr>
</tbody>
</table>
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.

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, 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”.

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

Duplicate rows

The pointer to the text or image data uniquely identifies each row. Therefore, a table that contains text or image data cannot contain duplicate rows unless all text and image data is NULL. If this is the case, the pointer has not been initialized.
User-defined datatypes

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

User-defined datatypes

Function
User-defined datatypes are built from the system datatypes and from the sysname user-defined datatype. 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.

Creating frequently used datatypes in the model database
A user-defined datatype must be created in each database in which it will be used. It is a good practice to 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.

Creating a user-defined datatype
Adaptive Server allows you to create user-defined datatypes, based on any system datatype, with the sp_addtype system procedure. 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 datatype is an exception to this rule. Though sysname is a user-defined datatype, you can use it 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.

### Renaming a user-defined datatype

Use `sp_rename` to rename a user-defined datatype.

### Dropping 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.

### Getting help on datatypes

Use the `sp_help` system procedure 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.

Topics covered are:

<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>52</td>
</tr>
<tr>
<td>Datatype conversion functions</td>
<td>58</td>
</tr>
<tr>
<td>Date functions</td>
<td>66</td>
</tr>
<tr>
<td>Mathematical functions</td>
<td>67</td>
</tr>
<tr>
<td>Security functions</td>
<td>69</td>
</tr>
<tr>
<td>String functions</td>
<td>70</td>
</tr>
<tr>
<td>System functions</td>
<td>71</td>
</tr>
<tr>
<td>Text and image functions</td>
<td>73</td>
</tr>
</tbody>
</table>

Types of functions

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

**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/time information.</td>
</tr>
<tr>
<td>Date functions</td>
<td>Do computations on datetime, smalldatetime, date and time values and their components, date parts.</td>
</tr>
<tr>
<td>Mathematical functions</td>
<td>Return values commonly needed for operations on mathematical data.</td>
</tr>
<tr>
<td>Security functions</td>
<td>Return security-related information.</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>Return special information from the database.</td>
</tr>
<tr>
<td>Text and image functions</td>
<td>Supply values commonly needed for operations on text and image data.</td>
</tr>
</tbody>
</table>

Table 2-2 lists the functions in alphabetical order.

### Table 2-2: List of Transact-SQL functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Type</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs</td>
<td>Mathematical</td>
<td>The absolute value of an expression.</td>
</tr>
<tr>
<td>acos</td>
<td>Mathematical</td>
<td>The angle (in radians) whose cosine is specified.</td>
</tr>
<tr>
<td>ascii</td>
<td>String</td>
<td>The ASCII code for the first character in an expression.</td>
</tr>
<tr>
<td>asin</td>
<td>Mathematical</td>
<td>The angle (in radians) whose sine is specified.</td>
</tr>
<tr>
<td>atan</td>
<td>Mathematical</td>
<td>The angle (in radians) whose tangent is specified.</td>
</tr>
<tr>
<td>atn2</td>
<td>Mathematical</td>
<td>The angle (in radians) whose sine and cosine are specified.</td>
</tr>
<tr>
<td>avg</td>
<td>Aggregate</td>
<td>The numeric average of all (distinct) values.</td>
</tr>
<tr>
<td>ceiling</td>
<td>Mathematical</td>
<td>The smallest integer greater than or equal to the specified value.</td>
</tr>
<tr>
<td>char</td>
<td>String</td>
<td>The character equivalent of an integer.</td>
</tr>
<tr>
<td>charindex</td>
<td>String</td>
<td>Returns an integer representing the starting position of an expression.</td>
</tr>
<tr>
<td>char_length</td>
<td>String</td>
<td>The number of characters in an expression.</td>
</tr>
<tr>
<td>col_length</td>
<td>System</td>
<td>The defined length of a column.</td>
</tr>
<tr>
<td>col_name</td>
<td>System</td>
<td>The name of the column whose table and column IDs are specified.</td>
</tr>
<tr>
<td>compare</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 char_expression1 is greater than char_expression2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 0 – indicates that char_expression1 is equal to char_expression2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- -1 – indicates that char_expression1 is less than char_expression2</td>
</tr>
<tr>
<td>convert</td>
<td>Datatype</td>
<td>The specified value, converted to another datatype or a different datetime</td>
</tr>
<tr>
<td></td>
<td>Conversion</td>
<td>display format.</td>
</tr>
<tr>
<td>cos</td>
<td>Mathematical</td>
<td>The cosine of the specified angle (in radians).</td>
</tr>
<tr>
<td>cot</td>
<td>Mathematical</td>
<td>The cotangent of the specified angle (in radians).</td>
</tr>
<tr>
<td>count</td>
<td>Aggregate</td>
<td>The number of (distinct) non-null values.</td>
</tr>
<tr>
<td>current_date</td>
<td>Date</td>
<td>Returns the current date.</td>
</tr>
<tr>
<td>current_time</td>
<td>Date</td>
<td>Returns the current time.</td>
</tr>
<tr>
<td>curunreservedpgs</td>
<td>System</td>
<td>The number of free pages in the specified disk piece.</td>
</tr>
<tr>
<td>data_pgs</td>
<td>System</td>
<td>The number of pages used by the specified table or index.</td>
</tr>
<tr>
<td>data_length</td>
<td>System</td>
<td>The actual length, in bytes, of the specified column or string.</td>
</tr>
<tr>
<td>datediff</td>
<td>Date</td>
<td>The date produced by adding a given number of years, quarters, hours, or</td>
</tr>
<tr>
<td></td>
<td></td>
<td>other date parts to the specified date.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The difference between two date expressions.</td>
</tr>
</tbody>
</table>
### Chapter 2  Transact-SQL Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Type</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>datename</td>
<td>Date</td>
<td>The name of the specified part of a date expression.</td>
</tr>
<tr>
<td>datepart</td>
<td>Date</td>
<td>The integer value of the specified part of a date expression.</td>
</tr>
<tr>
<td>day</td>
<td>Date</td>
<td>Returns an integer that represents the day in the datepart of a specified date.</td>
</tr>
<tr>
<td>db_id</td>
<td>System</td>
<td>The ID number of the specified database.</td>
</tr>
<tr>
<td>db_name</td>
<td>System</td>
<td>The name of the database whose ID number is specified.</td>
</tr>
<tr>
<td>degrees</td>
<td>Mathematical</td>
<td>The size, in degrees, of an angle with a specified number of radians.</td>
</tr>
<tr>
<td>derived_stat</td>
<td>System</td>
<td>Returns derived statistics for the specified object and index.</td>
</tr>
<tr>
<td>difference</td>
<td>String</td>
<td>The difference between two soundex values.</td>
</tr>
<tr>
<td>exp</td>
<td>Mathematical</td>
<td>The value that results from raising the constant e to the specified power.</td>
</tr>
<tr>
<td>floor</td>
<td>Mathematical</td>
<td>The largest integer that is less than or equal to the specified value.</td>
</tr>
<tr>
<td>get_appcontext</td>
<td>Security</td>
<td>Returns the value of the attribute in a specified context.</td>
</tr>
<tr>
<td>getdate</td>
<td>Date</td>
<td>The current system date and time.</td>
</tr>
<tr>
<td>hextoint</td>
<td>Datatype Conversion</td>
<td>The platform-independent integer equivalent of the specified hexadecimal string.</td>
</tr>
<tr>
<td>host_id</td>
<td>System</td>
<td>Returns the client computer’s operating system process ID for the current Adaptive Server client.</td>
</tr>
<tr>
<td>host_name</td>
<td>System</td>
<td>The current host computer name of the client process.</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>isnull</td>
<td>System</td>
<td>Substitutes the value specified in expression2 when expression1 evaluates to NULL..</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>String</td>
<td>The specified expression, trimmed of leading blanks.</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 uppercase equivalent of the specified expression.</td>
</tr>
<tr>
<td>max</td>
<td>Aggregate</td>
<td>The highest value in a column.</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>min</td>
<td>Aggregate</td>
<td>The lowest value in a column.</td>
</tr>
<tr>
<td>mut_excl_roles</td>
<td>System</td>
<td>The mutual exclusivity between two roles.</td>
</tr>
<tr>
<td>newid</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</td>
<td>System</td>
<td>Retrieves the next identity value that is available for the next insert.</td>
</tr>
<tr>
<td>object_id</td>
<td>System</td>
<td>The object ID of the specified object.</td>
</tr>
<tr>
<td>object_name</td>
<td>System</td>
<td>The name of the object whose object ID is specified.</td>
</tr>
<tr>
<td>pagesize</td>
<td>Mathematical</td>
<td>Returns the page size, in bytes, for the specified object.</td>
</tr>
<tr>
<td>patindex</td>
<td>String, Text and Image</td>
<td>The starting position of the first occurrence of a specified pattern.</td>
</tr>
<tr>
<td>pi</td>
<td>Mathematical</td>
<td>The constant value 3.1415926535897936.</td>
</tr>
<tr>
<td>power</td>
<td>Mathematical</td>
<td>The value that results from raising the specified number to a given power.</td>
</tr>
<tr>
<td>proc_role</td>
<td>System</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>ptn_data_pgs</td>
<td>System</td>
<td>The number of data pages used by a partition.</td>
</tr>
<tr>
<td>radians</td>
<td>Mathematical</td>
<td>The size, in radians, of an angle with a specified number of degrees.</td>
</tr>
<tr>
<td>rand</td>
<td>Mathematical</td>
<td>A random value between 0 and 1, generated using the specified seed value.</td>
</tr>
<tr>
<td>replicate</td>
<td>String</td>
<td>A string consisting of the specified expression repeated a given number of times.</td>
</tr>
<tr>
<td>reserved_pgs</td>
<td>System</td>
<td>The number of pages allocated to the specified table or index.</td>
</tr>
<tr>
<td>reverse</td>
<td>String</td>
<td>The specified string, with characters listed in reverse order.</td>
</tr>
<tr>
<td>right</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</td>
<td>Security</td>
<td>Removes a specific application context, or all application contexts.</td>
</tr>
<tr>
<td>role_contain</td>
<td>System</td>
<td>1 if role2 contains role1.</td>
</tr>
<tr>
<td>role_id</td>
<td>System</td>
<td>The system role ID of the role whose name you specify.</td>
</tr>
<tr>
<td>role_name</td>
<td>System</td>
<td>The name of a role whose system role ID you specify.</td>
</tr>
<tr>
<td>round</td>
<td>Mathematical</td>
<td>The value of the specified number, rounded to a given number of decimal places.</td>
</tr>
<tr>
<td>rowcnt</td>
<td>System</td>
<td>An estimate of the number of rows in the specified table.</td>
</tr>
<tr>
<td>rtrim</td>
<td>String</td>
<td>The specified expression, trimmed of trailing blanks.</td>
</tr>
<tr>
<td>set_appcontext</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</td>
<td>System</td>
<td>The login’s currently active roles.</td>
</tr>
<tr>
<td>show_sec_services</td>
<td>Security</td>
<td>A list of the user’s currently active security services.</td>
</tr>
<tr>
<td>sign</td>
<td>Mathematical</td>
<td>The sign (+1 for positive, 0, or -1 for negative) of the specified value.</td>
</tr>
<tr>
<td>sin</td>
<td>Mathematical</td>
<td>The sine of the specified angle (in radians).</td>
</tr>
</tbody>
</table>
sortkey
  System  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.

soundex
  String  A 4-character code representing the way an expression sounds.

space
  String  A string consisting of the specified number of single-byte spaces.

square
  Mathematical  Returns the square of a specified value expressed as a float.

sqrt
  Mathematical  The square root of the specified number.

str
  String  The character equivalent of the specified number.

str_replace
  String  Replaces any instances of the second string expression that occur within the first string expression with a third expression.

stuff
  String  The string formed by deleting a specified number of characters from one string and replacing them with another string.

substring
  String  The string formed by extracting a specified number of characters from another string.

sum
  Aggregate  The total of the values.

suser_id
  System  The server user’s ID number from the syslogins system table.

suser_name
  System  The name of the current server user, or the user whose server user ID is specified.

syb_quit

syb_sendmsg
  Sends a message to a User Datagram Protocol (UDP) port.

tan
  Mathematical  The tangent of the specified angle (in radians).

tempdb_id

textptr
  Text and Image  The pointer to the first page of the specified text column.

textvalid
  Text and Image  1 if the pointer to the specified text column is valid; 0 if it is not.

to_unichar
  String  A unichar expression having the value of the integer expression.

tsequal
  System  Compares timestamp values to prevent update on a row that has been modified since it was selected for browsing.

uhighsurr
  String  1 if the Unicode value at position start is the high half of a surrogate pair (which should appear first in the pair); otherwise 0.

ulowsurr
  String  1 if the Unicode value at position start is the low half of a surrogate pair (which should appear second in the pair); otherwise 0.

upper
  String  The uppercase equivalent of the specified string.

uscalar
  String  The Unicode scalar value for the first Unicode character in an expression.

used_pgs
  System  The number of pages used by the specified table and its clustered index.

user
  System  The name of the current server user.

user_id
  System  The ID number of the specified user or the current user.
Aggregate functions

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**
- **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 (12).

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 now have semantics that include the unichar data type.

---

### Function Details

<table>
<thead>
<tr>
<th>Function</th>
<th>Type</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_name</td>
<td>System</td>
<td>The name within the database of the specified user or the current user.</td>
</tr>
<tr>
<td>valid_name</td>
<td>System</td>
<td>0 if the specified string is not a valid identifier; a number other than 0 if the string is valid.</td>
</tr>
<tr>
<td>valid_user</td>
<td>System</td>
<td>1 if the specified ID is a valid user or alias in at least one database on this Adaptive Server.</td>
</tr>
<tr>
<td>year</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 `ansi_null_off` option is set (the default), there is no warning when an aggregate function encounters a null. If `ansi_null_on` 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)
Aggregate functions

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.

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
type
---------  --------  ---------
UNDECIDED  NULL    NULL
business   13.73   6,281.25
mod_cook   11.49   7,500.00
popular_comp 21.48  7,500.00
psychology 13.50   4,255.00
trad_cook   15.96   6,333.33
(6 rows affected)
```

### 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 put into a single group.

```sql
select type, title_id, avg(price), avg(advance)
from titles
group by type
order by type
type  title_id
--------  -------
UNDECIDED  MC3026 NULL    NULL
business   BU1032 13.73   6,281.25
business   BU1111 13.73   6,281.25
business   BU2075 13.73   6,281.25
business   BU7832 13.73   6,281.25
mod_cook   MC2222 11.49   7,500.00
mod_cook   MC3021 11.49   7,500.00
popular_comp PC1035 21.48   7,500.00
popular_comp PC8888 21.48   7,500.00
```
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>10.95</td>
<td>2,275.00</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>sum</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).

Aggregate functions cannot be used 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:
Aggregate functions

Start of select statement

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

where:

- 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:

```
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)

```
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>
</tbody>
</table>
```
CHAPTER 2  Transact-SQL Functions

```
<table>
<thead>
<tr>
<th>type</th>
<th>price</th>
<th>advance</th>
</tr>
</thead>
<tbody>
<tr>
<td>trad_cook</td>
<td>11.95</td>
<td>4,000.00</td>
</tr>
<tr>
<td>trad_cook</td>
<td>14.99</td>
<td>8,000.00</td>
</tr>
<tr>
<td>trad_cook</td>
<td>20.95</td>
<td>7,000.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sum</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>47.89</td>
<td>19,000.00</td>
</tr>
</tbody>
</table>
```

(7 rows affected)

```
<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>
</tbody>
</table>
```

Compute Result:

```
<table>
<thead>
<tr>
<th>type</th>
<th>price</th>
<th>advance</th>
</tr>
</thead>
<tbody>
<tr>
<td>trad_cook</td>
<td>11.95</td>
<td>4,000.00</td>
</tr>
<tr>
<td>trad_cook</td>
<td>14.99</td>
<td>8,000.00</td>
</tr>
<tr>
<td>trad_cook</td>
<td>20.95</td>
<td>7,000.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sum</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>47.89</td>
<td>19,000.00</td>
</tr>
</tbody>
</table>
```

(7 rows affected)

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:

```
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
```

Reference Manual: Building Blocks 57
Datatype conversion functions

Compute Result:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>17</td>
</tr>
</tbody>
</table>

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 statements that include `compute` generate tables that include the summary results, which are not stored in the database.

Datatype conversion functions

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

- `convert()`
- `inttohex()`
- `hextoint()`

The datatype conversion functions can be used 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.

Table 2-3 indicates whether individual datatype conversions are performed implicitly, explicitly, or are not supported.

**Table 2-3: Explicit, implicit, and unsupported datatype conversions**

<table>
<thead>
<tr>
<th>From</th>
<th>tinyint</th>
<th>smallint</th>
<th>int</th>
<th>decimal</th>
<th>numeric</th>
<th>float</th>
<th>real</th>
<th>[n]char</th>
<th>[n]varchar</th>
<th>unichar</th>
<th>univarchar</th>
<th>text</th>
<th>smallmoney</th>
<th>money</th>
<th>bit</th>
<th>smalldatetime</th>
<th>datetime</th>
<th>binary</th>
<th>varbinary</th>
<th>image</th>
</tr>
</thead>
<tbody>
<tr>
<td>tinyint</td>
<td>–</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>U</td>
<td>I</td>
<td>I</td>
<td>U</td>
<td>I</td>
<td>U</td>
<td>I</td>
<td>I</td>
<td>U</td>
</tr>
<tr>
<td>smallint</td>
<td>I</td>
<td>–</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>U</td>
<td>I</td>
<td>I</td>
<td>U</td>
<td>I</td>
<td>I</td>
<td>U</td>
<td>I</td>
<td>U</td>
</tr>
<tr>
<td>int</td>
<td>I</td>
<td>I</td>
<td>–</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>U</td>
<td>I</td>
<td>I</td>
<td>U</td>
<td>I</td>
<td>I</td>
<td>U</td>
<td>I</td>
<td>U</td>
</tr>
<tr>
<td>decimal</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>U</td>
<td>I</td>
<td>I</td>
<td>U</td>
<td>I</td>
<td>I</td>
<td>U</td>
<td>I</td>
<td>U</td>
</tr>
<tr>
<td>numeric</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>U</td>
<td>I</td>
<td>I</td>
<td>U</td>
<td>I</td>
<td>I</td>
<td>U</td>
<td>I</td>
<td>U</td>
</tr>
<tr>
<td>real</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>U</td>
<td>I</td>
<td>I</td>
<td>U</td>
<td>I</td>
<td>I</td>
<td>U</td>
<td>I</td>
<td>U</td>
</tr>
<tr>
<td>float</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>U</td>
<td>I</td>
<td>I</td>
<td>U</td>
<td>I</td>
<td>I</td>
<td>U</td>
<td>I</td>
<td>U</td>
</tr>
<tr>
<td>unichar</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>univarchar</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>text</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>smallmoney</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>money</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>bit</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>smalldatetime</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>datetime</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>binary</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</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 arithabortionumeric_truncation is on; implicit conversion allowed otherwise.
- U – unsupported conversion.
- – Conversion of a datatype to itself. These conversions are allowed, but are meaningless.
Datatype conversion functions

Converting character data to a non-character type

Character data can be converted to a non-character 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, SQL 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/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 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:

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

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

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

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:

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

-----------

5
Datatype conversion functions

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/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 2-6 on page 96 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

Data can be converted from one numeric type to another. If the new type is an exact numeric whose precision or scale is not sufficient to hold the data, errors can occur.

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.

**Note** The arithabort and arithignore options have been redefined for release 10.0 or later. If you use these options in your applications, examine them to be sure they are still producing the desired behavior.

### 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. The optional arith_overflow keyword can be omitted without any effect.
Datatype conversion functions

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 arithabort numeric_truncation option to determine how serious such an error is considered. 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.

Note For entry level ANSI SQL compliance, set:
- arithabort arith_overflow off
- arithabort 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 and 256 on machines that consider byte 0 least significant.

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.

### 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:

```sql
select convert(numeric (38, 18), 0x000000000000000006b14bd1e6eea0000000000000000000000000000000)
```

```
-----------
123.456000
```

This example converts the same numeric data back to binary:

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

```
---------------------------------------------------------------------------------------------------
0x000000000000000006b14bd1e6eea00000000000000000000000000000
```

### 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 the NULL to NOT NULL and NOT NULL to NULL.

**Date functions**

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

Date functions can be used in the select list or `where clause` of a query.

Use the `datetime` datatype only for dates after January 1, 1753. `datetime` 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 and “Date and time datatypes” for more information.

Adaptive Server automatically converts between character and `datetime` values when necessary (for example, when you compare a character value to a `datetime` 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 (2079 for <code>smalldatetime</code>)</td>
</tr>
<tr>
<td>quarter</td>
<td>qq</td>
<td>1 – 4</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.

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 adding datetime data. You can use the time datatype for time information.

### 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:

- abs
Mathematical functions

- acos
- asin
- atan
- atn2
- ceiling
- cos
- cot
- degrees
- exp
- floor
- lockscheme
- log
- log10
- pagesize
- pi
- power
- radians
- rand
- round
- sign
- sin
- sqrt
- tan

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:
Transact-SQL Functions

- `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.

**Note** The `arithabort` and `arithignore` options have been redefined for release 10.0 or later. If you use these options in your applications, examine them to be sure they still produce the desired effects.

Security functions

Security functions return security-related information.

The security functions are:

- `is_sec_service_on`
- `show_sec_services`
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
- stuff
- substring
- to_unichar
- uh highsurr
- ulowsurr
- upper
- uscalar

String functions can be nested, and they can be used 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.

**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, SQL 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_pgs`
- `datalength`
- `db_id`
- `db_name`
System functions

- host_id
- host_name
- index_col
- isnull
- lct_admin
- mut_excl_roles
- object_id
- object_name
- proc_role
- ptn_data_pgs
- reserved_pgs
- role_contain
- role_id
- role_name
- rowcnt
- show_role
- suser_id
- suser_name
- tsequal
- used_pgs
- 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 and image functions

Text and image functions operate on text and image 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 or image data that is retrieved by a `select` statement.

The `patindex` text function can be used on text and image columns and can also be considered a text and image function.

Use the `datalength` function to get the length of data in text and image columns.

Text and image columns cannot be used:

- As parameters to stored procedures
- 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
## `abs`

**Description**

Returns the absolute value of an expression.

**Syntax**

```
abs(numeric_expression)
```

**Parameters**

`numeric_expression`

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

**Examples**

Returns the absolute value of -1:

```
select abs(-1)
```

```
-----------
   1
```

**Usage**

- `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.

**Standards**

ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**

Any user can execute `abs`.

**See also**

“Mathematical functions” on page 67 for general information about mathematical functions.

**Functions**

`ceiling`, `floor`, `round`, `sign`
acos

Description
Returns the angle (in radians) whose cosine is specified.

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 whose cosine is 0.52:

```
select acos(0.52)
```

-------------------
1.023945

Usage
- acos, a mathematical function, returns the angle (in radians) whose 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 67 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
```sql
select au_lname, ascii(au_lname) from authors
where ascii(au_lname) < 70
```

<table>
<thead>
<tr>
<th>au_lname</th>
<th>ascii(au_lname)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bennet</td>
<td>66</td>
</tr>
<tr>
<td>Blotchet-Halls</td>
<td>66</td>
</tr>
<tr>
<td>Carson</td>
<td>67</td>
</tr>
<tr>
<td>DeFrance</td>
<td>68</td>
</tr>
<tr>
<td>Dull</td>
<td>68</td>
</tr>
</tbody>
</table>

Returns the authors last names and the ASCII codes for the first letters in their last names, if the ASCII code is less than 70.

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 70.

Functions
char, to_unichar
### asin

**Description**  
Returns the angle (in radians) whose sine is specified.

**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) whose sine is the specified value.

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

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

**See also**  
“Mathematical functions” on page 67 for general information about mathematical functions.

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

Description
Returns the angle (in radians) whose tangent is specified.

Syntax
atan(tangent )

Parameters
   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.

Examples
   select atan(0.50)
          ---------
          0.463648

Usage
   • atan, a mathematical function, returns the angle (in radians) whose tangent
     is the specified value.

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

Permissions
   Any user can execute atan.

See also
   “Mathematical functions” on page 67 for general information about
   mathematical functions.

Functions
   atn2, degrees, radians, tan
### atn2

<table>
<thead>
<tr>
<th>Description</th>
<th>Returns the angle (in radians) whose sine and cosine are specified.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>atn2(sine, cosine)</td>
</tr>
<tr>
<td>Parameters</td>
<td><strong>sine</strong> 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.</td>
</tr>
<tr>
<td></td>
<td><strong>cosine</strong> 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.</td>
</tr>
<tr>
<td>Examples</td>
<td><code>select atn2(.50, .48)</code></td>
</tr>
<tr>
<td></td>
<td><code>--------------------</code></td>
</tr>
<tr>
<td></td>
<td><code>0.805803</code></td>
</tr>
<tr>
<td>Usage</td>
<td>• atn2, a mathematical function, returns the angle (in radians) whose sine and cosine are specified.</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 atn2.</td>
</tr>
<tr>
<td>See also</td>
<td>“Mathematical functions” on page 67 for general information about mathematical functions.</td>
</tr>
</tbody>
</table>

**Functions** atn, degrees, radians, tan
**avg**

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

**Syntax**
\[
\text{avg}([\text{all} \mid \text{distinct}] \text{ expression})
\]

**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 249.

**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:

\[
\begin{align*}
\text{select } & \text{avg(advance), sum(total_sales)} \\
& \text{from titles} \\
& \text{where type = "business"}
\end{align*}
\]

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6,281.25</td>
<td>30788</td>
</tr>
</tbody>
</table>

**Example 2** Used with a `group by` clause, the aggregate functions produce single values for each group, rather than for the whole table. This statement produces summary values for each type of book:

\[
\begin{align*}
\text{select type, avg(advance), sum(total_sales)} \\
& \text{from titles} \\
& \text{group by type}
\end{align*}
\]

<table>
<thead>
<tr>
<th>type</th>
<th></th>
<th></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:
CHAPTER 2  Transact-SQL Functions

```
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 integer data, Adaptive Server treats the result as an int value, even if the datatype of the column is smallint or tinyint. To avoid overflow errors in DB-Library programs, declare all variables for results of averages or sums as type `int`.

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

- Since the average value is only defined on numeric datatypes, use with 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 52.

**Functions**  max, min
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 whose datatype is exact numeric, approximate numeric, money, or any type that can be implicitly converted to one of these types.

Examples
Example 1
```sql
select ceiling(123.45)
124
```

Example 2
```sql
select ceiling(-123.45)
-123
```

Example 3
```sql
select ceiling(1.2345E2)
24.000000
```

Example 4
```sql
select ceiling(-1.2345E2)
-123.000000
```

Example 5
```sql
select ceiling($123.45)
124.00
```

Example 6
```sql
select discount, ceiling(discount) from salesdetail
where title_id = "PS3333"
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 67.

**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**

```sql
select char(42)
```

```output

* `A`
```

**Example 2**

```sql
select xxx = char(65)
```

```output

---

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:

```sql
/* 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
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 70.

Functions  ascii, str
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 data, binary or
varbinary.

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

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

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 char_expr or uchar_expr is NULL, returns NULL.
• 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
70.

Function
patindex
### char_length

**Description**
Returns the number of characters in an expression.

**Syntax**
```
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**
```
select char_length(notes) from titles
where title_id = "PC9999"
--------------
39
```

**Example 2**
```
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)
-------------- --------------- ---------------
4              8               20
```

**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 multi-byte character sets, the number of characters in the expression is usually less 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 70.

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

Permissions
Any user can execute `char_length`.

See also
Function `datalength`
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:

```sql
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 71.
• To find the actual length of the data stored in each row, use datalength.
• For text 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 whose table and column IDs are specified.

**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)
-----------------------------
title
```

**Usage**
- **col_name**, a system function, returns the column’s name.
- For general information about system functions, see “System functions”
on page 71.

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

**Permissions**
Any user can execute **col_name**.

**See also**
- **Functions**  
  - db_id, object_id
CHAPTER 2  Transact-SQL Functions

**compare**

**Description**

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

**Syntax**

```sql
compare ({char_expression1|uchar_expression1},
          {char_expression2|uchar_expression2}),
          [{collation_name | collation_ID}]
```

**Parameters**

- `char_expression1` or `uchar_expression1` are the character expressions you want to compare to `char_expression2` or `uchar_expression2`.
- `char_expression2` or `uchar_expression2` are the character expressions against which you want to compare `char_expression1` or `uchar_expression1`.
- `char_expression1` and `char_expression2` can be one of the following:
  - 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 one of the following:
  - 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 2-5 shows the valid values.
- `collation_ID` is an integer constant or a variable that specifies the collation to use. Table 2-5 shows the valid values.

**Examples**

**Example 1** Compares aaa and bbb:

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

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

```
1> select compare (("aaa"),("bbb"))
2> go
-----------
-1
(1 row affected)
```

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

```
1> select compare ("aaa","bbb","binary")
2> go
-----------
-1
(1 row affected)
```

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

```
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 6 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. 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:
If this occurs, Adaptive Server issues a warning message, but the query or transaction that contained the compare function continues to run.

- Both char_expression1, uchar_expression1, and char_expression2 and uchar_expression2 must be characters that are encoded in the server’s default character set.

- Either char_expression1, uchar_expression1, 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 a 0 value.
  - If char_expression1 or uchar_expression1 is empty, the function returns a -1.

The compare function does not equate empty strings and strings containing only spaces, as does. 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 will not compare equally.

- If either char_expression1, uchar_expression1; or char_expression2, uchar_expression2 is NULL, then the result will be 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 2-5 lists the valid values for `collation_name` and `collation_ID`.

### Table 2-5: Collation names and IDs

<table>
<thead>
<tr>
<th>Description</th>
<th>Collation name</th>
<th>Collation ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary sort</td>
<td>binary</td>
<td>50</td>
</tr>
<tr>
<td>Default Unicode multilingual</td>
<td>default</td>
<td>0</td>
</tr>
<tr>
<td>CP 850 Alternative no accent</td>
<td>altnoacc</td>
<td>39</td>
</tr>
<tr>
<td>CP 850 Alternative lower case first</td>
<td>altdict</td>
<td>45</td>
</tr>
<tr>
<td>CP 850 Alternative no case preference</td>
<td>altnocsp</td>
<td>46</td>
</tr>
<tr>
<td>CP 850 Scandinavian dictionary</td>
<td>scandict</td>
<td>47</td>
</tr>
<tr>
<td>CP 850 Scandinavian no case preference</td>
<td>scannocp</td>
<td>48</td>
</tr>
<tr>
<td>GB Pinyin</td>
<td>gbpiyin</td>
<td>n/a</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 Cyrillic dictionary</td>
<td>cyrdict</td>
<td>63</td>
</tr>
<tr>
<td>ISO 8859-5 Russian dictionary</td>
<td>rUSDICT</td>
<td>58</td>
</tr>
<tr>
<td>ISO 8859-9 Turkish dictionary</td>
<td>turdict</td>
<td>72</td>
</tr>
<tr>
<td>Shift-JIS binary order</td>
<td>sjisbin</td>
<td>259</td>
</tr>
<tr>
<td>Thai dictionary</td>
<td>thaidict</td>
<td>1</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
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 data type, 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 2-6 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 styles numbers 1 through 7 (101 through 107) or 10 through 12 (110 through 112) in Table 4-4 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 will reflect 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 will be displayed.

### Table 2-6: Display formats for date/time information

<table>
<thead>
<tr>
<th>Symbolic value</th>
<th>Datatype</th>
<th>Datetime</th>
<th>Date</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>0 or 100</td>
<td>mm/dd/yyyy</td>
<td>mm/dd/yy</td>
<td>00:00:00:000PM(AM)</td>
</tr>
<tr>
<td>1</td>
<td>101</td>
<td>mm/dd/yyyy</td>
<td>mm/dd/yy</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>102</td>
<td>yy/mm/dd</td>
<td>yy/mm/dd</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>103</td>
<td>dd/mm/yy</td>
<td>dd/mm/yy</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>104</td>
<td>dd.mm.yy</td>
<td>dd.mm.yy</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>105</td>
<td>dd-mm-yy</td>
<td>dd-mm-yy</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>106</td>
<td>dd.mm.yy</td>
<td>dd mm yy</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>107</td>
<td>mon dd, yy</td>
<td>mon dd, yy</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>108</td>
<td>hh:mm:ss</td>
<td>hh:mm:ss</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>109</td>
<td>mm dd yy</td>
<td>mm dd yyyy</td>
<td>hh:mm:ss/zzzAM(PM)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>hh:mm:ss/zzzAM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>110</td>
<td>mm-dd-yy</td>
<td>mm-dd-yy</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>111</td>
<td>yy/mm/dd</td>
<td>yy/mm/dd</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>112</td>
<td>yymmdd</td>
<td>yymmdd</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>113</td>
<td>yy/dd/mm</td>
<td>yy/dd/mm</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>114</td>
<td>mm/yy/dd</td>
<td>mm/yy/dd</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>115</td>
<td>dd/yy/mm</td>
<td>dd/yy/mm</td>
<td></td>
</tr>
</tbody>
</table>
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 $1.11:

```sql
select convert(bit, $1.11)
```

**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)
```
Adaptive Server Enterprise

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 58.
- `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).
- 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.
- Unichar expressions can be used as a destination data type or they can be converted to another data type. Unichar expressions can be converted either explicitly between any other data type supported by the server, or implicitly.
- If length is not specified 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 either 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 will be truncated leaving only the date portion. If a time value is converted to a `datetime` value, a default date portion of Jan 1, 1900 will be 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 will be 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 the 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 67.

Standards
ANSI SQL – Compliance level: Transact-SQL extension

Permissions
Any user can execute cos.

See also
Functions acos, degrees, radians, sin
**cot**

<table>
<thead>
<tr>
<th>Description</th>
<th>Returns the cotangent of the specified angle.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td><code>cot(angle)</code></td>
</tr>
<tr>
<td>Parameters</td>
<td><code>angle</code> is any approximate numeric (float, real, or double precision) column name, variable, or constant expression.</td>
</tr>
<tr>
<td>Examples</td>
<td><code>select cot(90)</code></td>
</tr>
<tr>
<td></td>
<td><code>--------------------</code></td>
</tr>
<tr>
<td></td>
<td><code>--------------------</code></td>
</tr>
<tr>
<td></td>
<td><code>-0.501203</code></td>
</tr>
<tr>
<td>Usage</td>
<td>• <code>cot</code>, a mathematical function, returns the cotangent of the specified angle, in radians.</td>
</tr>
<tr>
<td></td>
<td>• For general information about mathematical functions, see “Mathematical functions” on page 67.</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 <code>cot</code>.</td>
</tr>
<tr>
<td>See also</td>
<td><strong>Functions</strong> degrees, radians, sin</td>
</tr>
</tbody>
</table>
count

Description Returns the number of (distinct) non-null values or the number of selected rows.

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 249.

Examples

**Example 1** Finds the number of different cities in which authors live:

```sql
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:

```sql
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 52.

- 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).

• count() can be used 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
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
Used to find the current date as it exists on the server.

Standards
ANSI SQL – Entry level Compliance.

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

<table>
<thead>
<tr>
<th>Description</th>
<th>Returns the current time.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>current_time()</td>
</tr>
<tr>
<td>Parameters</td>
<td>None.</td>
</tr>
</tbody>
</table>

**Examples**

**Example 1** Finds the current time:

```
1> select current_date()
2> go
--------------------------
Aug 29 2003
(1 row affected)
```

**Example 2** Use with datename:

```
1> select datename(minute, current_time())
2> go
--------------------------
45
(1 row affected)
```

**Usage**

Used to find the current time as it exists on the server

**Standards**

ANSI SQL – Entry level Compliance.

**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:

```
select db_name(dbid), d.name, curunreservedpgs(dbid, lstart, unreservedpgs)
from sysusages u, sysdevices d
where d.low <= u.size + vstart
    and d.high >= u.size + vstart -1
    and d.status &2 = 2
```

```
master master 184
master master 832
tempdb master 464
tempdb master 1016
tempdb master 768
model master 632
sybsystemprocs master 1024
pubs2 master 248
```

Example 2

Displays the number of free pages on the segment for dbid starting on sysusages.lstart:

```
select curunreservedpgs (dbid, sysusages.lstart, 0)
```

Usage

- 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 71.

- If the database is open, the value is taken from memory; if the database is not in use, the value is taken from the unreservedpgs column in sysusages.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute curunreservedpgs.
See also Functions db_id, lct_admin
**data_pgs**

**Description**
Returns the number of pages used by the specified table or index.

**Syntax**
```
data_pgs([dbid], object_id, [data_oam_pg_id | index_oam_pg_id])
```

**Parameters**
- **dbid**
  is the dbid 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.

- **data_oam_pg_id**
  is the page ID for a data OAM page, stored in the doampg column of sysindexes.

- **index_oam_pg_id**
  is the page ID for an index OAM page, stored in the ioampg column of sysindexes.

**Examples**

**Example 1**
Estimates the number of data pages used by user tables (which have object IDs that are greater than 100). An indid of 0 indicates a table without a clustered index; an indid of 1 indicates a table with a clustered index. This example does not include nonclustered indexes or text chains:

```
select sysobjects.name,
   Pages = data_pgs(sysindexes.id, doampg)
from sysindexes, sysobjects
where sysindexes.id = sysobjects.id
   and sysindexes.id > 100
   and (indid = 1 or indid = 0)
```

**Example 2**
Estimates the number of data pages used by user tables (which have object IDs that are greater than 100), nonclustered indexes, and page chains:

```
select sysobjects.name,
   Pages = data_pgs(sysindexes.id, ioampg)
from sysindexes, sysobjects
where sysindexes.id = sysobjects.id
   and sysindexes.id > 100
   and (indid > 1)
```

**Usage**
- `data_pgs`, a system function, returns the number of pages used by a table (doampg) or index (ioampg). You must use this function in a query run against the sysindexes table. For more information on system functions, see “System functions” on page 71.
• data_pgs works only on objects in the current database.

• The result does not include pages used for internal structures. To see a report of the number of pages for the table, clustered index, and internal structures, use used_pgs.

Accuracy of results

• If used on the transaction log (syslogs), the result may not be accurate and can be off by up to 16 pages.

Errors

• Instead of returning an error, data_pgs returns 0 if any of the following are true:
  • The object_id does not exist in sysobjects
  • The control_page_id does not belong to the table specified by object_id
  • The object_id is -1
  • The page_id is -1

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute data_pgs.

See also

Functions  object_id, rowcnt

System procedure  sp_spaceused
**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. It can be of any datatype. expression 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:

```sql
select Length = datalength(pub_name)
from publishers
```

<table>
<thead>
<tr>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>20</td>
</tr>
</tbody>
</table>

**Usage**
- datalength, a system function, returns the length of expression in bytes.
- 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 their 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**
Returns the date produced by adding a given number of years, quarters, hours, or other date parts to the specified date.

**Syntax**
`dateadd(date_part, integer, date expression)`

**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 66.
- `numeric` is an integer expression.
- `date expression` is an expression of type `datetime`, `smalldatetime`, `date`, `time`, or a character string in a datetime format.

**Examples**

**Example 1** 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 2** Add one day to a date:

```sql
declare @a date
select @a = "apr 12, 9999"
select dateadd(dd, 1, @a)
```

**Example 3** Add five minutes to a time:

```sql
select dateadd(mi, 5, convert(time, "14:20:00"))
```

**Example 4** Add 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)
```

**Example 5** Although there are limits for each `date_part`, as with `datetime` values, higher values can be added resulting in the values rolling over to the next significant field:

```sql
--Add 24 hours to a datetime
```
**dateadd**

```sql
select dateadd(hh, 24, "4/1/1979")
--------------------------
Apr 2 1979 12:00AM

--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 66.

- `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 it is meaningful only if the result date returned by `dateadd` changes by at least one minute.

- 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 58.

  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>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>1900-2079 (smalldatetime)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0001-9999 (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>1054</td>
</tr>
<tr>
<td>Day</td>
<td>dd</td>
<td>1-7</td>
</tr>
<tr>
<td>dayofyear</td>
<td>dy</td>
<td>1-366</td>
</tr>
</tbody>
</table>

112 Adaptive Server Enterprise
<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</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>
</tbody>
</table>

**Standards**

ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**

Any user can execute `dateadd`.

**See also**

<table>
<thead>
<tr>
<th>Datatypes</th>
<th>Date and time datatypes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commands</td>
<td>select, where clause</td>
</tr>
<tr>
<td>Functions</td>
<td>datediff, datename, datepart, getdate</td>
</tr>
</tbody>
</table>
datediff

Description
Returns the difference between two dates.

Syntax
datediff(datepart, date expression1, date expression2)

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 66.

date expression1
is an expression of type datetime, smalldatetime, date, time, or a character string in a datetime format.

date expression2
is an expression of type datetime, smalldatetime, date, time, or a character string in a datetime format.

Examples

Example 1 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 2 Find 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 3 Find 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 4 Find 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)
```
Example 5  Overflow size of milliseconds return value:

```sql
select datediff(ms, convert(date, "4/1/1753"), convert(date, "4/1/9999"))
```

Msg 535, Level 16, State 0:
Line 2:
Difference of two datetime fields caused overflow at runtime.
Command has been aborted

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 66.
- **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 $date_2 - date_1$, 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 get 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 **smalldatetime** values are used, 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 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.
  - 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 either a date, time, datetime, or smalldatetime value as its second argument.

Syntax  

\[ \text{datename (datepart, date expression)} \]

Parameters  

- \text{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 66.

- \text{date expression}  
  is an expression of type datetime, smalldatetime, date, time, or a character string in a datetime format.

Examples  

**Example 1**  Assumes a current date of November 20, 2000:

\[ \begin{align*}
  \text{select datename(m, getdate())} \\
  \text{November}
\end{align*} \]

**Example 2**  Find the month name of a date:

\[ \begin{align*}
  \text{declare @a date} \\
  \text{select @a = "apr 12, 0001"} \\
  \text{select datename(mm, @a)} \\
  \text{-------------------------------} \\
  \text{April}
\end{align*} \]

**Example 3**  Find the seconds of a time:

\[ \begin{align*}
  \text{declare @a time} \\
  \text{select @a = "20:43:22"} \\
  \text{select datename(ss, @a)} \\
  \text{-------------------------------} \\
  \text{22}
\end{align*} \]

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 66.

- 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.
datename

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 (the second argument) as an integer. Takes either a date, time, datetime, or smalldatetime value as its second argument. If the datepart is hour, minute, second, or millisecond, the result is zero.

**Syntax**
```
datepart(date_part, date expression)
```

**Parameters**
- `date_part` is a date part. Table 2-8 lists the date parts, the abbreviations recognized by datepart, and the acceptable 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.

  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.

**date expression**

is an expression of type datetime, smalldatetime, date, time, or a character string in a datetime format.

**Examples**

**Example 1** This example assumes a current date of November 25, 1995:

```sql
select datepart(month, getdate())
--
11
```
Example 2

```sql
select datepart(year, pubdate) from titles where type = "trad_cook"
```

```
--------
1990
1985
1987
```

Example 3

```sql
select datepart(cwk,'1993/01/01')
```

```
--------
53
```

Example 4

```sql
select datepart(cyr,'1993/01/01')
```

```
--------
1992
```

Example 5

```sql
select datepart(cdw,'1993/01/01')
```

```
--------
5
```

Example 6  Find the hours in a time:

```sql
declare @a time
select @a = "20:43:22"
select datepart(hh, @a)
```

```
--------
20
```

Example 7  If a hour, minute, or second portion is requested from a date using datename() or datepart() the result is the default time, zero. If a month, day, or year is requested from a time using datename() or datepart() the result is the default date, Jan 1 1900:

```sql
--Find the hours in a date
declare @a date
select @a = "apr 12, 0001"
select datepart(hh, @a)
```

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

```sql
--Find the month of a time
```
```sql
declare @a time
select @a = "20:43:22"
select datename(mm, @a)
-------------------------------
January
```

When a null value is given to a datetime function as a parameter, null will be 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 66.

- `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 `calweekofyear`, `calyearofweek`, or `caldayorweek`, the date returned may be different for the same unit of time. For example, if Adaptive Server is configured to use US English as the default language, the following returns 1988:

  ```sql
datepart(cyr, "1/1/1989")
```

However, the following returns 1989:

  ```sql
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 US English as their default language, the first day of the week as 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. The default behavior can be changed on a per-session basis with `set datefirst`. See the `datefirst` option of the `set` command for more information.

- `calweekofyear`, 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 be abbreviated as `cdw`, returns the ordinal position of the day within the week. You cannot use `calweekofyear`, `calyearofweek`, and `caldayofweek` as date parts for `dateadd`, `datediff` and `datename`.

122  Adaptive Server Enterprise
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.

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

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

**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:

```sql
  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_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

    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 71.

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

Permissions
Any user can execute db_id.

See also
Functions db_name, object_id
db_name

Description
Returns the name of the database whose ID number is specified.

Syntax
\textit{db\_name}([database\_id])

Parameters
- \textit{database\_id} is a numeric expression for the database ID (stored in \textit{sysdatabases.dbid}).

Examples
\textbf{Example 1} Returns the name of the current database:
\begin{verbatim}
select db_name()
\end{verbatim}

\textbf{Example 2}
\begin{verbatim}
select db_name(4)
\end{verbatim}

\textbf{sybsystemprocs}

Usage
- \textit{db\_name}, a system function, returns the database name.
- If no \textit{database\_id} is supplied, \textit{db\_name} returns the name of the current database.
- For general information about system functions, see “System functions” on page 71.

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

Permissions
Any user can execute \textit{db\_name}.

See also
Functions \textit{col\_name, db\_id, object\_name}
### 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.

  When money datatypes are used, internal conversion to float may cause loss of precision.

- For general information about mathematical functions, see “Mathematical functions” on page 67.

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

**Permissions**
Any user can execute degrees.

**See also**
- Functions  
radians
**derived_stat**

**Description**

Returns derived statistics for the specified object and index.

**Syntax**

```
derived_stat({ object_name | object_id }, { index_name | index_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. This 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

- **statistic**
  the derived statistic to be returned. Available statistics are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>data page cluster ratio or dpcr</td>
<td>The data page cluster ratio for the object/index pair</td>
</tr>
<tr>
<td>index page cluster ratio or ipcr</td>
<td>The index page cluster ratio for the object/index pair</td>
</tr>
<tr>
<td>data row cluster ratio or drcr</td>
<td>The data row cluster ratio for the object/index pair</td>
</tr>
<tr>
<td>large io efficiency or lgio</td>
<td>The large io efficiency for the object/index pair</td>
</tr>
<tr>
<td>space utilization or sput</td>
<td>The space utilization for the object/index pair</td>
</tr>
</tbody>
</table>

**Examples**

**Example 1** Selects the space utilization for the titleidind index of the titles table:

```
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":

```
select derived_stat("titles", 2, "dpcr")
```

**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.
Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Only the table owner can execute derived_stat.

See also

Documents  Performance and Tuning Guide for:

- “Access Methods and Query Costing for Single Tables”
- “Statistics Tables and Displaying Statistics with optdiag”

Utilities  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**

```sql
select difference("smithers", "smothers")
```

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

**Example 2**

```sql
select difference("smothers", "brothers")
```

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

**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 `char_expr1`, `uchar_expr1`, or `char_expr2`, `uchar_expr2` is NULL, returns 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).

- For general information about string functions, see “String functions” on page 70.

**Standards**

ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**

Any user can execute difference.

**See also**

Functions 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 67.

**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, or int), 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 67.

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. 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 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
```

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 ACF.
For more information on the Application Context Facility 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
getcode()

Parameters
None.

Examples

Example 1  Assumes a current date of November 25, 1995, 10:32 a.m.:

```
select getdate()
```

```
Nov 25 1995 10:32AM
```

Example 2  Assumes a current date of November:

```
select datepart(month, getdate())
```

```
1
```

Example 3  Assumes a current date of November:

```
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 66.

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
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. This must be either a
character type column or 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:

```sql
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 58.

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

Permissions
Any user can execute hextoint.

See also
Functions convert, inttohex
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()
----------------------------- -----------------------
ephemeris 2309
```

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 70.

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 71.

Standards  ANSI SQL – Compliance level: Transact-SQL extension.
Permissions  Any user can execute host_name.
See also  Function host_id
**identity_burn_max**

**Description**
Tracks the identity burn max value for a given table. This function only returns the value and does not do an update.

**Syntax**
`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. This function only returns the value and does not do an update.

**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.

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 70.

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

Permissions
Any user can execute index_col.

See also
Functions object_id
System procedures 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 70.

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

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

---

142  Adaptive Server Enterprise
### inttohex

**Description**
Returns the platform-independent hexadecimal equivalent of the specified integer.

**Syntax**
```
inttohex (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 58.

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

**Permissions**
Any user can execute inttohex.

**See also**
- **Functions** `convert`, `hextoint`
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 70.

- 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
is_sec_service_on

Description
Returns 1 if the security service is active and 0 if it is not.

Syntax
is_sec_service_on(security_service_nm)

Parameters
security_service_nm
is the name of the security service.

Examples
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, run this query:

```sql
select * from syssecmechs
```

The result might look something like:

```
sec_mech_name available_service
----------------- ------------------
dce unifiedlogin

dce mutualauth

dce delegation

dce integrity

dce confidentiality

dce detectreplay

dce detectseq
```

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
**lct_admin**

**Description**
Manages the last-chance threshold.

Returns the current value of the last-chance threshold.

Aborts transactions in a transaction log that has reached its last-chance threshold.

**Syntax**
```
lct_admin(["lastchance" | "logfull" | "reserved_for_rollbacks"],
   database_id
   |"reserve", (log_pages | 0 )
   | "abort", process-id [, database-id])
```

**Parameters**
- **lastchance**
  creates a last-chance threshold in the specified database.

- **logfull**
  returns 1 if the last-chance threshold 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 last-chance threshold 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 last-chance threshold.

- **0**
  returns the current value of the last-chance threshold. The size of the last-chance threshold 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 last-chance threshold 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
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
the ID of a database whose transaction log 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:

        select lct_admin("lastchance", 1)

Example 2 Returns 1 if the last-chance threshold for the database with db_id
of 6 has been crossed, and 0 if it has not:

        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:

        select lct_admin("reserve", 64)

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

Example 4 Returns the current last-chance threshold of the transaction log in
the database from which the command was issued:

        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:

        select lct_admin("abort", 83)

Example 6 Aborts all open transactions in the database with database ID 5.
This form awakens any processes that may be suspended at the log segment
last-chance threshold:

        select lct_admin("abort", 0, 5)

Example 7 Determines the number of pages reserved for rollbacks in the
pubs2 database, which has a pubid of 5:

        select lct_admin("reserved_for_rollbacks", 5, 0)

Example 8 Describes the free space available for a database with database ID
of 4:
**select lct_admin("logsegment_freepages", 4)**

**Usage**
- `lct_admin`, a system function, manages the log segment's last-chance threshold. For general information about system functions, see “String functions” on page 70.
- 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**
- **Documents**  *System Administration Guide.*
- **Command**  `dump transaction`
- **Function**  `curunreservedpgs`
- **System procedures**  `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 the `substring` on page 215.

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, or 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**
```sql
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 Adaptive Server Java 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:
```sql
select license_enabled("ase_dtm")
```
```
--------------
1
```

**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*
- **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
Shows the results when a user without appropriate permissions attempts to list the application contexts.

```sql
select list_appcontext ([context_name])
Context Name: (CONTEXT1)
Attribute Name: (ATTR1) Value: (VALUE2)
Context Name: (CONTEXT2)
Attribute Name: (ATTR1) Value: (VALUE1)

select list_appcontext()
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 ACF.

See also
For more information on the Application Context Facility 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)
```

Or
```
lockscheme(object_id [, db_id])
```

**Parameters**
- **object_name**
  The name of the object whose locking scheme this function returns. `object_name` can also be a fully qualified name.
- **db_id**
  The ID of the database specified by `object_id`.
- **object_id**
  The ID of the object whose locking scheme this function 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 (note that the `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`.
  - You do not provide a `db_id`.
  - You 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`. 

---

| 154 | Adaptive Server Enterprise |
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 67.

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     \[\text{log10}(\text{approx\_numeric})\]

Parameters     \textit{approx\_numeric}

is any approximate numeric (float, real, or double precision) column name, variable, or constant expression.

Examples     

\[
\begin{align*}
\text{select } \text{log10}(20) \\
\hline
1.301030
\end{align*}
\]

Usage     

- \textit{log10}, a mathematical function, returns the base 10 logarithm of the specified value.

- For general information about mathematical functions, see “Mathematical functions” on page 67.

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

Permissions     Any user can execute \textit{log10}.

See also     **Functions**  \(\text{log}, \text{power}\)
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

```sql
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 70.

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

Permissions Any user can execute `lower`.

See also Functions `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

```sql
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 lower-case Unicode equivalent of the specified expression. Characters in the expression that have no lower-case equivalent are left unmodified.
- For general information about string functions, see “String functions” on page 70.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute `ltrim`.

See also

`rtrim`
max

Description
Returns the highest value in an expression.

Syntax
\texttt{max}(\textit{expression})

Parameters
\textit{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 \textit{discount} column of the \textit{salesdetail} table as a new column:

\begin{center}
\texttt{select max(discount) from salesdetail}
\end{center}

\begin{center}
\texttt{-------------------}
\end{center}

\begin{center}
62.200000
\end{center}

Example 2 Returns the maximum value in the \textit{discount} column of the \textit{salesdetail} table as a new row:

\begin{center}
\texttt{select discount from salesdetail compute max(discount)}
\end{center}

Usage

- \texttt{max}, an aggregate function, finds the maximum value in a column or expression. For general information about aggregate functions, see “Aggregate functions” on page 52.

- \texttt{max} can be used with exact and approximate numeric, character, and \textit{datetime} columns. It cannot be used with \textit{bit} columns. With character columns, \texttt{max} finds the highest value in the collating sequence. \texttt{max} ignores null values. \texttt{max} implicitly converts \texttt{char} datatypes to \texttt{varchar}, \texttt{unichar} datatypes to \texttt{univarchar}, stripping all trailing blanks.

- \texttt{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 \texttt{max} when there is an index on the aggregated column, unless:
  - The \textit{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 \texttt{where} clause

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

Permissions
Any user can execute \texttt{max}. 
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 249.

**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 52.
- `min` can be used with numeric, character, time and datetime columns. It cannot be used with bit columns. With character columns, `min` finds the lowest value in the sort sequence. `min` implicitly converts `char` datatypes to `varchar`, `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**
`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.
- For general information about system functions, see “System functions” on page 71.

Standards
ANSI SQL – Compliance level: Transact-SQL extension

Permissions
Any user can execute mut_excl_roles.

See also
- Commands alter role, create role, drop role, grant, set, revoke
- Functions proc_role, role_contain, role_id, role_name
- System procedures sp_active roles, sp_display roles, 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, 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 and includes dashes.
- -0x0 – returns the GUID as a varbinary.

Examples
Example 1 Creates a table with varchar columns 32 bytes long and 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
```

Example 2 Produces a GUID that includes dashes.

```sql
select newid(1)
go
```

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
strtobin(newid())
go
```
sp_bindefault default_guid, "t.UUID"
go
insert t (UUID_VC) values (newid())
go

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. Any other value passed to `newid` generates GUIDs with dashes and is more easily readable.

- `newid` can be used 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 `strtobin` function to convert the GUID value to 16-byte binary data. However, using `strtobin` 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`. 

Reference Manual: Building Blocks 165
### 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.
object_id

Description: Returns the object ID of the specified object.

Syntax: 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

```
select object_id("titles")
```

```
-----------
208003772
```

Example 2

```
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.
- For general information about system functions, see “System functions” on page 71.

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

Permissions: Any user can execute object_id.

See also: Functions col_name, db_id, object_name

System procedure sp_help
**object_name**

**Description**  
Returns the name of the object whose object ID is specified.

**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**

```sql
select object_name(208003772)

-----------------------------
titles
```

**Example 2**

```sql
select object_name(1, 1)

-----------------------------
sysobjects
```

**Usage**
- `object_name`, a system function, returns the object’s name.
- For general information about system functions, see “System functions” on page 71.

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

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

**See also**  
*Functions*  
`col_name`, `db_id`, `object_id`

*System procedures*  
`sp_help`
pagesize

Description
Returns the page size, in bytes, for the specified object.

Syntax
pagesize(object_name [, index_name])
Or:
pagesize(object_id [, db_id [, index_id]])

Parameters
object_name
the name of the object whose page size this function returns.

index_name
indicates the name of the index whose pagesize you want returned.

object_id
the ID of the object whose page size this function returns.

db_id
the ID of the database in which the object with object_name resides.

index_id
the ID of the index whose page size you want returned.

Examples

Example 1 Selects the pagesize for the title_id index in the current database.

select pagesize("title", "title_id")

Example 2 The following 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 last example defaults to the current database):

select pagesize(1234,2, null)
selct pagesize(1234,2)
selct pagesize(1234)

Example 3 The following all default to the current database:

select pagesize(1234, null, 2)
selct pagesize(1234)

Example 4 Selects the pagesize for the titles table (object_id 224000798) from the pubs2 database (db_id 4):

select pagesize(224000798, 4)

Example 5 Returns the pagesize for the non-clustered 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")`) or 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 zero.

- If the specified object does not exist, `pagesize` returns NULL.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute `pagesize`. 
patindex

Description
Returns the starting position of the first occurrence of a specified pattern.

Syntax
patindex('%pattern%', char_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 that can be used in pattern, see “Pattern matching with wildcard
characters” on page 265.

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.

cchars 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:

```
select au_id, patindex('%circus%', copy)
from blurbs
```

```
au_id
--------
486-29-1786 0
648-92-1872 0
998-72-3567 38
899-46-2035 31
672-71-3249 0
409-56-7008 0
```

Example 2

```
select au_id, patindex('%circus%', copy,
```
**patindex**

```sql
using chars)
from blurbs

Example 3  The same as Example 1:

```sql
select au_id, patindex("%circus%", copy,
using chars)
from blurbs
```

Example 4  Finds all the rows in `sysobjects` that start with "sys" and whose fourth character is “a”, “b”, “c”, or “d”:

```sql
select name
from sysobjects
where patindex("sys[a-d]%", name) > 0
```

```
name
------------------------
sysalternates
sysattributes
syscharsets
syscolumns
syscomments
sysconfigures
sysconstraints
syscurconfigs
sysdatabases
sysdepends
sysdevices
```

**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 zero if `pattern` is not found.
- `patindex` can be used on all character data, including `text` and `image` data.
- 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`, returns 0.
- 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).
For general information about string functions, see “String functions” on page 70.

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

```sql
select pi()

-------------------
3.141593
```

Usage

- `pi`, a mathematical function, returns the constant value of 3.1415926535897931.

- For general information about mathematical functions, see “Mathematical functions” on page 67.

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

Permissions Any user can execute `pi`.

See also **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`.

For expressions of type numeric or decimal, 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 67.

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

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

**See also**
- **Functions** `exp`, `log`, `log10`
**proc_role**

**Description**
Returns information about whether the user has been granted the specified 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**
- `proc_role`, a system function, checks whether an invoking user has been granted, and has activated, the specified role.
- `proc_role` returns 0 if any of the following are true:
  - the user has not been granted the specified role
  - the user has not been granted a role which contains the specified role
  - the user has 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.
- For general information about system functions, see “System functions” on page 71.

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

**Permissions**
Any user can execute `proc_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
**ptn_data_pgs**

**Description**
Returns the number of data pages used by a partition.

**Syntax**

```sql
ptn_data_pgs(object_id, partition_id)
```

**Parameters**

- `object_id`
  is the object ID for a table, stored in the `id` column of `sysobjects`, `sysindexes`, and `syspartitions`.

- `partition_id`
  is the partition number of a table.

**Examples**

```sql
select ptn_data_pgs(object_id("salesdetail"), 1)
```

```
5
```

**Usage**

- `ptn_data_pgs`, a system function, returns the number of data pages in a partitioned table.
- Use the `object_id` function to get an object’s ID, and use `sp_helpartition` to list the partitions in a table.
- The data pages returned by `ptn_data_pgs` may be inaccurate. Use the `update partition statistics`, `dbcc checktable`, `dbcc checkdb`, or `dbcc checkalloc` commands before using `ptn_data_pgs` to get the most accurate value.
- For general information about system functions, see “System functions” on page 71.

**Standards**

ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**
Only the table owner can execute `ptn_data_pgs`.

**See also**

- **Commands**
  - `dbcc`, `update partition statistics`
- **Functions**
  - `data_pgs`, `object_id`
- **System procedures**
  - `sp_helpartition`
radians

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

select radians(2578)

-----------

  44

Usage

• radians, a mathematical function, converts degrees to radians. Results are
  of the same type as numeric.

  For expressions of type numeric or decimal, the results have an internal
  precision of 77 and a scale equal to that of the numeric expression.

  When money datatypes are used, internal conversion to float may cause
  loss of precision.

• For general information about mathematical functions, see “Mathematical
  functions” on page 67.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute radians.

See also

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 pseudo-random 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 pseudo-random 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.
- For general information about mathematical functions, see “Mathematical functions” on page 67.

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

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

**See also**
- **Datatypes** Approximate numeric datatypes
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

```sql
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 will fit into a 16K-space, whichever is less.

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

- For general information about string functions, see “String functions” on page 70.

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

Permissions
Any user can execute `replicate`.

See also
- **Functions** stuff
**reserved_pgs**

**Description**
Returns the number of pages allocated to the specified table or index, and reports pages used for internal structures.

**Syntax**
```sql
reserved_pgs(object_id, {doampg | ioampg})
```

**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`.
- `doampg | ioampg` specifies table (`doampg`) or index (`ioampg`).

**Examples**
Returns the page count for the `syslogs` table:
```sql
select reserved_pgs(id, doampg)
from sysindexes where id =
object_id("syslogs")
```

**Usage**
- `reserved_pgs`, a system function:
  - Returns the number of pages allocated to a table or an index
  - Reports pages used for internal structures
  - Works only on objects in the current database
  - For general information about system functions, see “System functions” on page 71.

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

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

**See also**
- **Commands** `update statistics`
- **Functions** `data_pgs`
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, returns NULL.
• Surrogate pairs are treated as indivisible and are not reversed.
• For general information about string functions, see “String functions” on page 70.

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

Permissions
Any user can execute reverse.

See also
Functions lower, upper
right

Description
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)
---
cde
```

**Example 2**
```sql
select right("abcde", 2)
--
de
```

**Example 3**
```sql
select right("abcde", 6)
-----
abcde
```

**Example 4**
```sql
select right(0x12345000, 3)
-------
0x345000
```

**Example 5**
```sql
select right(0x12345000, 2)
------
0x5000
```

**Example 6**
```sql
select right(0x12345000, 6)
---------
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, returns NULL.

- For general information about string functions, see “String functions” on page 70.

Standards

ANSI SQL – Compliance level: Transact-SQL extension

Permissions

Any user can execute `right`.

See also

**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:

```sql
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:

```sql
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 Application Context Facility 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.
• For more information about system functions, see “System functions” on page 71.

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

Permissions
Any user can execute role_contain.

See also
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 role whose 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`.
- For more information about system functions, see “System functions” on page 71.

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

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

**See also**
- **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 role whose system role ID you specify.

Syntax
role_name(role_id)

Parameters
role_id
is the system role ID (srid) of the role. Role names are stored in syssrvroles.

Examples
select role_name(01)
--------------------------
ssr_role

Usage
• role_name, a system function, returns the role name.

• For more information about system functions, see “System functions” on page 71.

Standards
ANSI SQL – Compliance level: Transact-SQL extension

Permissions
Any user can execute role_name.

See also
Functions mut_excl_roles, proc_role, role_contain, role_id


round

Description
Returns the value of the specified number, rounded to a given number of decimal places.

Syntax
round(number, decimal_places)

Parameters
number
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.

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 decimal_places determines the number of significant digits to the right of the decimal point; a negative decimal_places, 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

- `round` always returns a value. If `decimal_places` is negative and exceeds the number of significant digits in `number`, Adaptive Server returns a result of 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:

  `select round(55.55, -3)`

- For general information about mathematical functions, see “Mathematical functions” on page 67.

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

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

**See also**
- **Functions**
  - `abs`, `ceiling`, `floor`, `sign`, `str`
rowcnt

Description
Returns an estimate of the number of rows in the specified table.

Syntax
rowcnt(sysindexes.doampg)

Parameters
sysindexes.doampg is the row count maintained in sysindexes.

Examples
```sql
select name, rowcnt(sysindexes.doampg)
from sysindexes
where name in
  (select name from sysobjects where type = "U")
```

name
-------------------------------
roysched 87
salesdetail 116
stores 7
discounts 4
au_pix 0
blurbs 6

Usage
- rowcnt, a system function, returns the estimated number of rows in a table.
- The value returned by rowcnt can vary unexpectedly when Adaptive Server reboots and recovers transactions. The value is most accurate after running one of the following commands:
  - dbcc checkalloc
  - dbcc checkdb
  - dbcc checktable
  - update all statistics
  - update statistics
- For general information about system functions, see “System functions” on page 71.

Standards
ANSI SQL – Compliance level: Transact-SQL extension

Permissions
Any user can execute rowcnt.

See also
Catalog stored procedures sp_statistics
Commands dbcc, update all statistics, update statistics
Function data_pgs
System procedures  sp_helppartition, sp_spaceused
**rtrim**

<table>
<thead>
<tr>
<th>Description</th>
<th>Returns the specified expression, trimmed of trailing blanks.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>`rtrim(char_expr</td>
</tr>
<tr>
<td>Parameters</td>
<td><code>char_expr</code> is a character-type column name, variable, or constant expression of <code>char</code>, <code>varchar</code>, <code>nchar</code> or <code>nvarchar</code> type.</td>
</tr>
<tr>
<td></td>
<td><code>uchar_expr</code> is a character-type column name, variable, or constant expression of <code>unichar</code>, or <code>univarchar</code> type.</td>
</tr>
<tr>
<td>Examples</td>
<td><code>select rtrim(&quot;abcd &quot;)</code></td>
</tr>
<tr>
<td>Usage</td>
<td>• <code>rtrim</code>, a string function, removes trailing blanks.</td>
</tr>
<tr>
<td></td>
<td>• For Unicode, a blank is defined as the Unicode value U+0020.</td>
</tr>
<tr>
<td></td>
<td>• If <code>char_expr</code> or <code>uchar_expr</code> is NULL, returns NULL.</td>
</tr>
<tr>
<td></td>
<td>• Only values equivalent to the space character in the current character set are removed.</td>
</tr>
<tr>
<td></td>
<td>• For general information about string functions, see “String functions” on page 70.</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 <code>rtrim</code>.</td>
</tr>
<tr>
<td>See also</td>
<td><strong>Functions</strong> <a href="#">ltrim</a></td>
</tr>
</tbody>
</table>
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(2048).

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 the following:

    select set_appcontext("CONTEXT1", "ATTR1", "VALUE1")
    -------------------
    -1

Example 2 Shows set_appcontext including a datatype conversion in the value.

    declare@numericvarchar varchar(25)
    select @numericvar = "20"
    select set_appcontext ("CONTEXT1", "ATTR2",
    convert(char(20), @numericvar))
    -------------------
    0

Example 3 Shows the result when a user without appropriate permissions attempts to set the application context.

    select set_appcontext("CONTEXT1", "ATTR2", "VALUE1")
    -------------------
Usage

- This function 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. If you want to assign new values to a context, remove the context and re-create it with 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
For more information on the Application Context Facility 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.
- For general information about system functions, see “System functions” on page 71.

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

**Permissions**
Any user can execute show_role.

**See also**
- **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:
```
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, or int), 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.
• For general information about mathematical functions, see “Mathematical functions” on page 67.

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

Permissions
Any user can execute sign.

See also
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).
- For general information about mathematical functions, see “Mathematical functions” on page 67.

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

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

**See also**  
* 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 2-10 shows the valid values.

collation_ID
is an integer constant or a variable that specifies the collation to use. Table 2-10 shows the valid values.

Examples
Example 1 Shows sorting by European language dictionary order:

```sql
select * from cust_table where cust_name like "TI%" order by (sortkey(cust_name, "dict")
```

Example 2 Shows sorting by simplified Chinese phonetic order:

```sql
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:

```sql
select * from cust_table where cust_name like "TI%" order by cust_french_sort
```

Example 4 Shows sorting by Simplified Chinese phonetic order using pre-existing keys:

```sql
select * from cust_table where cust_name like "TI%" order by cust_chinese_sort.
```
CHAPTER 2 Transact-SQL Functions

Usage

- **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. When you want to retrieve the character data in the desired order, the select statement only needs to include 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 6 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>4K (4096 bytes)</td>
<td>4010</td>
<td>4008 bytes</td>
<td></td>
</tr>
<tr>
<td>8K (8192 bytes)</td>
<td>8106</td>
<td>8104 bytes</td>
<td></td>
</tr>
<tr>
<td>16K (16384 bytes)</td>
<td>16298</td>
<td>16296 bytes</td>
<td></td>
</tr>
<tr>
<td>DOL tables</td>
<td>2K (2048 bytes)</td>
<td>1964</td>
<td>1958 bytes</td>
</tr>
<tr>
<td>4K (4096 bytes)</td>
<td>4012</td>
<td>4006 bytes</td>
<td></td>
</tr>
<tr>
<td>8K (8192 bytes)</td>
<td>8108</td>
<td>8102 bytes</td>
<td></td>
</tr>
<tr>
<td>16K (16384 bytes)</td>
<td>16300</td>
<td>16294 bytes if table does not include any variable length columns</td>
<td></td>
</tr>
<tr>
<td></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>
<td></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.
• *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, the *unicode* default sort order is used.

• 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, you must regenerate the 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 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 higher than Adaptive Server version 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 at `$SYBASE/collate/unicode`. 

Adaptive Server Enterprise
Both of these methods work equally well, but a “built-in” table is tied to a Sybase Adaptive Server database, an external table is not. If you use an Adaptive Server database, a built-in table provides the best performance. Both of these methods can handle any mix of English, European, and Asian languages.

There are two ways of using 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. Note however, that this method generates sort keys on-the-fly, and therefore does not provide optimum performance on large datasets of over 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 out the list by executing either the stored procedure sp_helpsort, or by querying and selecting the name, id, and description from syscharsets (type is between 2003 and 2999).

- Table 2-10 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>Binary sort</td>
<td>binary</td>
<td>50</td>
</tr>
<tr>
<td>Default Unicode multilingual</td>
<td>default</td>
<td>0</td>
</tr>
<tr>
<td>CP 850 Alternative no accent</td>
<td>altnoacc</td>
<td>39</td>
</tr>
<tr>
<td>CP 850 Alternative lower case first</td>
<td>altdict</td>
<td>45</td>
</tr>
<tr>
<td>CP 850 Alternative no case preference</td>
<td>altnocsp</td>
<td>46</td>
</tr>
<tr>
<td>CP 850 Scandinavian dictionary</td>
<td>scandict</td>
<td>47</td>
</tr>
<tr>
<td>CP 850 Scandinavian no case preference</td>
<td>scannocsp</td>
<td>48</td>
</tr>
<tr>
<td>GB Pinyin</td>
<td>gbpinyin</td>
<td>n/a</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>
</tbody>
</table>
### sortkey

<table>
<thead>
<tr>
<th>Description</th>
<th>Collation name</th>
<th>Collation ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latin-1 Spanish no case</td>
<td>espnocs</td>
<td>56</td>
</tr>
<tr>
<td>Latin-1 Spanish no accent</td>
<td>esnoac</td>
<td>57</td>
</tr>
<tr>
<td>ISO 8859-5 Cyrillic dictionary</td>
<td>cyrdict</td>
<td>63</td>
</tr>
<tr>
<td>ISO 8859-5 Russian dictionary</td>
<td>rusdict</td>
<td>58</td>
</tr>
<tr>
<td>ISO 8859-9 Turkish dictionary</td>
<td>turdict</td>
<td>72</td>
</tr>
<tr>
<td>Shift-JIS binary order</td>
<td>sjisbin</td>
<td>259</td>
</tr>
<tr>
<td>Thai dictionary</td>
<td>thaidict</td>
<td>1</td>
</tr>
</tbody>
</table>

**Standards**

ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**

Any user can execute sortkey.

**See also**

**Functions**  compare
soundex

Description
Returns a 4-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 4-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 alpha 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.

• For general information about string functions, see “String functions” on page 70.

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

Permissions
Any user can execute soundex.

See also
Functions  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.

• For general information about string functions, see “String functions” on page 70.

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

Permissions
Any user can execute space.

See also
Functions isnull, ltrim
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:

```sql
select square(total_sales) from titles
```

```plaintext
-------------
16769025.00000
15023376.00000
350513284.00000
...
16769025.00000
(18 row(s) affected)
```

Example 2 Returns the square from a money column:

```sql
select square(price) from titles
```

```plaintext
-------------
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 Functions 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.
• For general information about mathematical functions, see “Mathematical functions” on page 67.

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

Permissions
Any user can execute sqrt.

See also
Functions  power
str

Description
Returns the character equivalent of the specified number.

Syntax
\[
\text{str(approx\_numeric [, length [, decimal]])}
\]

Parameters
- \text{approx\_numeric}
  is any approximate numeric (float, real, or double precision) column name, variable, or constant expression.
- \text{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.
- \text{decimal}
  sets the number of decimal digits to be returned. The default is 0.

Examples

Example 1
\[
\text{select str(1234.7, 4)}
\]
\[
----
1235
\]

Example 2
\[
\text{select str(-12345, 6)}
\]
\[
-------
-12345
\]

Example 3
\[
\text{select str(123.45, 5, 2)}
\]
\[
-----
123.5
\]

Usage
- \text{str}, a string function, returns a character representation of the floating point number. For general information about string functions, see “String functions” on page 70.
- \text{length} and \text{decimal} are optional. If given, they must be non-negative. \text{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, \text{str} returns a row of asterisks of the specified length. For example:
\[
\text{select str(123.456, 2, 4)}
\]
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 (string_expression2) that occur within the first string expression (string_expression1) with a third expression (string_expression3).

Syntax
replace("string_expression1", "string_expression2", "string_expression3")

Parameters
string_expression1
is the source string, or the string expression to be searched, expressed as char, varchar, unichar, univarchar, varbinary, or binary datatype.

string_expression2
is the pattern string, or the string expression to find within the first expression (string_expression1). string_expression2 is expressed as char, varchar, unichar, univarchar, varbinary, or binary datatype.

string_expression3
is the replacement string expression, expressed as char, varchar, unichar, univarchar, binary, or varbinary datatype.

Examples
Example 1 Replaces the string def within the string cdefghi with yyy.

    replace("cdefghi","def","yyy")
-----------------------------
cyyyghi
(1 row(s) affected)

Example 2 Replaces all spaces with "toyota".

    select str_replace("chevy, ford, mercedes", ",","toyota")
-----------------------------
chevy,toyotaford,toyotamercedes
(1 row(s) affected)

Note
Adaptive Server converts an empty string constant to a string of 1 space automatically, to distinguish the string from NULL values.

Usage
- Returns varchar data if string_expression (1,2, or 3) is char or varchar.
- Returns univar data if string_expression (1,2, or 3) is unichar or univarchar.
- Returns varbinary data if 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.
The result length may vary, depending upon what is known about the argument values when the expression is compiled. If all the arguments are variables with known constant values, Adaptive Server calculates the result length as:

\[ \text{result\_length} = ((s/p)*(r-p)+s) \]

where

- \( s \) = length of source string
- \( p \) = length of pattern string
- \( r \) = length of replacement string

if \((r-p) \leq 0\), result length = \( s \)

If the source string (\textit{string\_expression1}) is a column, and \textit{string\_expression2} and \textit{string\_expression3} 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.

\( \text{result\_len} \) never exceeds 16384.

**Standards**

ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**

Any user can execute \textit{str\_replace}.

**See also**

\textbf{Datatypes}  char, varchar, binary, varbinary, unichar, univarchar

\textbf{Functions}  length
CHAPTER 2  Transact-SQL Functions

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_expr2 at start. For general information about string functions, see “String functions” on page 70.

- If the start position or the length is negative, a NULL string is returned. If the start position is 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, returns NULL. If char_expr1 or or uchar_expr1 is a string value and char_expr2 or uchar_expr2 is NULL, replaces the deleted characters with nothing.

- 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 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 70.
• If any of the arguments to substring are NULL, substring returns NULL.
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 249.

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:

```sql
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 whole table. This statement produces summary values for each type of book:

```sql
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:

```sql
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 52.
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. To avoid overflow errors in DB-Library programs, declare all variables for results of averages or sums as type int.

- You cannot use sum with the binary datatypes.
- Since this function only defines 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 71.
- 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

```sql
suser_name([server_user_id])
```

Parameters

- `server_user_id` is an Adaptive Server user ID.

Examples

**Example 1**

```sql
select suser_name()
----------------------
sa
```

**Example 2**

```sql
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.

- For general information about system functions, see “System functions” on page 71.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute `suser_name`.

See also

**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.

```sql
select syb_quit()
-------------
CT-LIBRARY error:
  ct_results(): network packet layer:
  internal net library error: Net-Library operation terminated due to disconnect
```

**Usage**
syb_quit can be used 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 caution when using `syb_sendmsg` to send sensitive information across the network. By enabling this functionality, the user accepts any security problems which 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`
## 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).
- For general information about mathematical functions, see “Mathematical functions” on page 67.

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

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

**See also**
- **Functions** `atan`, `atn2`, `degrees`, `radians`
### tempdb_id

**Description**  
The `tempdb_id()` reports the temporary database that a given session is assigned to. 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, then `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, execute:

```sql
select spid from master..sysprocesses
where tempdb_id(spid) = db_id("tempdatabase")
```

**Usage**  
`select tempdb_id()` gives the same result as `select @@tempdbid`.

**Standards**

**Permissions**

**See also**  
*Commands*  
`select`
textptr

Description
Returns a pointer to the first page of a text or image 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 put into a local variable `@val` and supplied as a parameter to the `readtext` command, which returns 5 bytes, starting at the second byte (offset of 1):

```sql
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:

```sql
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 or an 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.
- For general information about text and image functions, see “Text and image functions” on page 73.

**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
- **Datatypes** text and image datatypes
- **Functions** `textvalid`
- **Commands** `insert`, `update`, `readtext`, `writetext`
**textvalid**

**Description**

Returns 1 if the pointer to the specified text 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:

```
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 73.

**Standards**

ANSI SQL – Compliance level: Transact-SQL extension.

**Permissions**

Any user can execute `textvalid`.

**See also**

- [Datatypes](#) text and image datatypes
- [Functions](#) `textptr`

---

226

Adaptive Server Enterprise
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, returns NULL.

• For general information about string functions, see “String functions” on page 70.

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

Permissions: Any user can execute to_unichar.

See also: Datatypes text and image datatypes

Functions: char
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, updates the row. If the values are not equal, returns an 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 71.

- 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

- The tsequal function should not be used 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 a timestamp column is used as a search clause, it should be compared 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 with alter table. For example, the following adds 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. For example:

```sql
update oldtable
set col1 = col1
```

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute tsequal.

See also

Datatypes  Timestamp datatype
uhighsurr

Returns 1 if the Unicode value at position \texttt{start} is the high half of a surrogate pair (which should appear first in the pair). Returns 0 otherwise.

\textbf{Syntax}

\texttt{uhighsurr(uchar\_expr, start)}

\textbf{Parameters}

- \texttt{uchar\_expr} is a character-type column name, variable, or constant expression of \texttt{unichar}, or \texttt{univarchar} type.
- \texttt{start} specifies the character position to investigate.

\textbf{Usage}

- \texttt{uhighsurr}, a string function, allows you to write explicit code for surrogate handling. Specifically, if a substring starts on a Unicode character where \texttt{uhighsurr()} is true, you need to extract a substring of at least 2 Unicode values. (\texttt{substr} will not extract half of a surrogate pair.)
- If \texttt{uchar\_expr} is NULL, returns NULL.
- For general information about string functions, see “String functions” on page 70.

\textbf{Standards}

ANSI SQL – Compliance level: Transact-SQL extension.

\textbf{Permissions}

Any user can execute \texttt{uhighsurr}.

\textbf{See also}

- \texttt{ulowsurr}
**ulowsurr**

**Description**
Returns 1 if the Unicode value at position `start` is the low half of a surrogate pair (which should appear second in the pair). Returns 0 otherwise.

**Syntax**
`ulowsurr(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**
- `ulowsurr`, a string function, allows you to write explicit code around adjustments performed by `substr()`, `stuff()`, and `right()`. Specifically, if a substring ends on a Unicode value where `ulowsurr()` is true, the user knows to extract a substring of 1 less characters (or 1 more). `substr()` does not extract a string that contains an unmatched surrogate pair.
- If `uchar_expr` is NULL, returns NULL.
- For general information about string functions, see “String functions” on page 70.

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

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

**See also**
Functions `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, 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.
- For general information about string functions, see “String functions” on page 70.

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

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

**See also**
- Functions: `lower`
| **uscalar** | 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, and error occurs and the operation is aborted.  
  • For general information about string functions, see “String functions” on page 70. |
| **Standards** | ANSI SQL – Compliance level: Transact-SQL extension. |
| **Permissions** | Any user can execute uscalar. |
| **See also** | Functions ascii |
used_pgs

Returns the number of pages used by a table or index. For an all-pages-locked table with a clustered index, it returns the sum of the table and index pages.

Syntax

used_pgs(object_id, doampg, ioampg)

Parameters

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.

doampg

is the page number for the object allocation map of a table or clustered index, stored in the doampg column of sysindexes.

ioampg

is the page number for the allocation map of a nonclustered index, stored in the ioampg column of sysindexes.

Examples

Example 1

Returns the number of pages used by the data and clustered index of the titles table:

```
select name, id, indid, doampg, ioampg
from sysindexes where id = object_id("titles")
name     id    indid    doampg   ioampg
---------- ----------- ------ -------- -------
titleind  208003772   1    560     552
        208003772   2     0     456
select used_pgs(208003772, 560, 552)
----------
6
```

Example 2

Returns the number of pages used by the stores table, which has no index:

```
select name, id, indid, doampg, ioampg
from sysindexes where id = object_id("stores")
name     id    indid    doampg   ioampg
---------- ----------- ------ -------- -------
stores    240003886   0    464     0
select used_pgs(240003886, 464, 0)
----------
2
```

Usage

• used_pgs, a system function, returns:
For all-pages-locked tables with a clustered index – the sum of the table and index pages

For data-only-locked tables and tables with no clustered index – the number of used pages in the table

For clustered and nonclustered indexes on data-only-locked tables – the number of pages in the index

In the examples, indid 0 indicates a table; indid 1 indicates a clustered index; an indid of 2–250 is a nonclustered index; and an indid of 255 is text or image data.

used_pgs only works on objects in the current database.

Each table and each index on a table has an object allocation map (OAM), which contains information about the number of pages allocated to and used by an object. This information is updated by most Adaptive Server processes when pages are allocated or deallocated. The sp_spaceused system procedure reads these values to provide quick space estimates. Some dbcc commands update these values while they perform consistency checks.

For general information about system functions, see “System functions” on page 71.

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

Permissions
Any user can execute used_pgs.

See also
Functions data_pgs, object_id
**user**

Description

Returns the name of the current user.

Syntax

user

Parameters

None.

Examples

```sql
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”.

- For general information about system functions, see “System functions” on page 71.

Standards

ANSI SQL – Compliance level: Transact-SQL extension.

Permissions

Any user can execute user.

See also

**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 71.
- **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. For general information about system functions, see “System functions” on page 71.
- 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**
* Functions: `user_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.

#### Syntax
`valid_name(character_expression)`

#### 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.

#### Examples
Creates a procedure to verify that identifiers are valid:

```sql
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 30 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 (@).
- For general information about system functions, see “System functions” on page 71.

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

#### Permissions
Any user can execute `valid_name`.

#### See also
- **System procedure** `sp_checkreswords`
valid_user

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.

• For general information about system functions, see “System functions” on page 71.

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

System procedures sp_addlogin, sp_adduser
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
Global Variables

Global variables are system-defined variables updated by Adaptive Server on an ongoing basis. For example, @@error contains the last error number generated by the system.

To view the value for any global variable, enter:

```
select variable_name
```

For example:

```
select @@char_convert
```

Topics covered are:

<table>
<thead>
<tr>
<th>Topics</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptive Server’s global variables</td>
<td>243</td>
</tr>
</tbody>
</table>

Adaptive Server’s global variables

The following are the global variables available for Adaptive Server:

<table>
<thead>
<tr>
<th>Global variable</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<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>
<tr>
<td>@@bulkbatchsize</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 Component Integration Services User’s Guide.</td>
</tr>
<tr>
<td>@@char_convert</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>@@cis_rpc_handling</td>
<td>Returns 0 if cis rpc handling is off. Returns 1 if cis rpc handling is on. For more information, see the Component Integration Services User’s Guide.</td>
</tr>
</tbody>
</table>
### Adaptive Server’s global variables

<table>
<thead>
<tr>
<th>Global variable</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>@@cis_version</td>
<td>Returns the date and version of Component Integration Services.</td>
</tr>
<tr>
<td>@@client_csexpansion</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>@@client_csid</td>
<td>Returns -1 if the client character set has never been initialized. Returns the client character set ID from syscharsets for the connection if the client character set has been initialized.</td>
</tr>
<tr>
<td>@@client_csname</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>@@cmpstate</td>
<td>Returns the current mode of Adaptive Server in a high availability environment.</td>
</tr>
<tr>
<td>@@connections</td>
<td>Returns the number of user logins attempted.</td>
</tr>
<tr>
<td>@@cpu_busy</td>
<td>Returns the number of seconds, in CPU time, that Adaptive Server’s CPU was performing Adaptive Server work.</td>
</tr>
<tr>
<td>@@curloid</td>
<td>Returns the current session’s lock owner ID.</td>
</tr>
<tr>
<td>@@datefirst</td>
<td>Set using <code>set datefirst n</code> where <code>n</code> is a value between 1 and 7. Returns the current value of@@datefirst, indicating the specified first day of each week, expressed as tinyint.</td>
</tr>
<tr>
<td></td>
<td>The default value in Adaptive Server is Sunday (based on the us_language default), which you set by specifying <code>set datefirst 7</code>. See the <code>datefirst</code> option of the <code>set</code> command for more information on settings and values.</td>
</tr>
<tr>
<td>@@dbts</td>
<td>Returns the timestamp of the current database.</td>
</tr>
<tr>
<td>@@error</td>
<td>Returns the error number most recently generated by the system.</td>
</tr>
<tr>
<td>@@errorlog</td>
<td>Returns the full path to the directory in which the Adaptive Server errorlog is kept, relative to $SYBASE directory (%SYBASE% on NT).</td>
</tr>
<tr>
<td>@@failedoverconn</td>
<td>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.</td>
</tr>
<tr>
<td>@@guestuserid</td>
<td>Returns the ID of the guest user.</td>
</tr>
<tr>
<td>@@haconnection</td>
<td>Returns the name of the companion server in a high availability setup.</td>
</tr>
<tr>
<td>@@haconnection</td>
<td>Returns a value greater than 0 if the connection has the failover property enabled. This is a session-specific property.</td>
</tr>
<tr>
<td>@@heapmemsize</td>
<td>Returns the size of the heap memory pool, in bytes. See the System Administration Guide for more information on heap memory.</td>
</tr>
<tr>
<td>@@identity</td>
<td>Returns the most recently generated IDENTITY column value.</td>
</tr>
<tr>
<td>@@idle</td>
<td>Returns the number of seconds, in CPU time, that Adaptive Server has been idle.</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 number of seconds in CPU time that Adaptive Server has spent doing input and output operations.</td>
</tr>
<tr>
<td>Global variable</td>
<td>Definition</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------</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>@@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>@@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>
<tr>
<td>@@mingroupid</td>
<td>Returns the lowest group user ID. The lowest value is 16384.</td>
</tr>
<tr>
<td>@@min_poolsize</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>@@minspid</td>
<td>Returns 1, which is the lowest value for spid.</td>
</tr>
<tr>
<td>@@minsuid</td>
<td>Returns the minimum server user ID. The lowest value is -32768.</td>
</tr>
<tr>
<td>@@minuserid</td>
<td>Returns the lowest user ID. The lowest value is -32768.</td>
</tr>
<tr>
<td>@@ncharsize</td>
<td>Returns the maximum length, in bytes, of a character set in the current server default character set.</td>
</tr>
<tr>
<td>@@nestlevel</td>
<td>Returns the current nesting level.</td>
</tr>
</tbody>
</table>
Adaptive Server's global variables

<table>
<thead>
<tr>
<th><strong>Global variable</strong></th>
<th><strong>Definition</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>@@nodeid</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>@@options</td>
<td>Returns a hexadecimal representation of the session’s set options.</td>
</tr>
<tr>
<td>@@packet_errors</td>
<td>Returns the number of errors detected by Adaptive Server while reading and writing packets.</td>
</tr>
<tr>
<td>@@pack_received</td>
<td>Returns the number of input packets read by Adaptive Server.</td>
</tr>
<tr>
<td>@@pack_sent</td>
<td>Returns the number of output packets written by Adaptive Server.</td>
</tr>
<tr>
<td>@@pagesize</td>
<td>Returns the server's virtual page size.</td>
</tr>
<tr>
<td>@@parallel_degree</td>
<td>Returns the current maximum parallel degree setting.</td>
</tr>
<tr>
<td>@@probesuid</td>
<td>Returns a value of 2 for the probe user ID.</td>
</tr>
<tr>
<td>@@procid</td>
<td>Returns the stored procedure ID of the currently executing procedure.</td>
</tr>
<tr>
<td>@@recovery_state</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>@@rowcount</td>
<td>Returns the number of rows affected by the last query. @@rowcount is set to 0 by any command that does not return rows, such as an if, update, or delete statement. With cursors, @@rowcount represents the cumulative number of rows returned from the cursor result set to the client, up to the last fetch request.</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>@@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_EXECUTEABLE, 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>
</tbody>
</table>
### Global Variables

<table>
<thead>
<tr>
<th>Global variable</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>@@stringsize</td>
<td>Returns the amount of character data returned from a <code>toString()</code> 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>@@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>
<tr>
<td>@@textdbid</td>
<td>Returns the database ID of a database containing an object with the column referenced by @@textptr.</td>
</tr>
<tr>
<td>@@textobjid</td>
<td>Returns the object ID of an object containing the column referenced by @@textptr.</td>
</tr>
<tr>
<td>@@textptr</td>
<td>Returns the text pointer of the last text or image column inserted or updated by a process (Not the same as the <code>textptr</code> function).</td>
</tr>
<tr>
<td>@@textptr_parameters</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>@@textsize</td>
<td>Returns the limit on the number of bytes of text or image data a <code>select</code> returns. Default limit is 32K bytes for <code>isql</code>; the default depends on the client software. Can be changed for a session with <code>set textsize</code>.</td>
</tr>
<tr>
<td>@@textts</td>
<td>Returns the text timestamp of the column referenced by @@textptr.</td>
</tr>
<tr>
<td>@@thresh_hysteresis</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>@@timeticks</td>
<td>Returns the number of microseconds per tick. The amount of time per tick is machine-dependent.</td>
</tr>
<tr>
<td>@@total_errors</td>
<td>Returns the number of errors detected by Adaptive Server while reading and writing.</td>
</tr>
<tr>
<td>@@total_read</td>
<td>Returns the number of disk reads by Adaptive Server.</td>
</tr>
<tr>
<td>@@total_write</td>
<td>Returns the number of disk writes by Adaptive Server.</td>
</tr>
<tr>
<td>@@tranchained</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>@@trancount</td>
<td>Returns the nesting level of transactions in the current user session.</td>
</tr>
<tr>
<td>@@transactional_rpc</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 <code>enable xact coordination</code> and <code>set option transactional_rpc</code> in the Reference Manual. Also, see the Component Integration Services User’s Guide.</td>
</tr>
<tr>
<td>@@transtate</td>
<td>Returns the current state of a transaction after a statement executes in the current user session.</td>
</tr>
<tr>
<td>@@unicharsize</td>
<td>Returns 2, the size of a character in unichar.</td>
</tr>
</tbody>
</table>
### Adaptive Server's global variables

<table>
<thead>
<tr>
<th>Global variable</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>@@version</td>
<td>Returns the date, version string, and so on of the current release of Adaptive Server.</td>
</tr>
<tr>
<td>@@version_as_integer</td>
<td>Returns the version of the current release of Adaptive Server as an integer.</td>
</tr>
</tbody>
</table>
Expressions, Identifiers, and Wildcard Characters

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>249</td>
</tr>
<tr>
<td>Identifiers</td>
<td>259</td>
</tr>
<tr>
<td>Pattern matching with wildcard characters</td>
<td>265</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>
<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>

Table 4-1: Types of expressions used in syntax statements

Reference Manual: Building Blocks 249
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 4-2: 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, 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  0</td>
</tr>
<tr>
<td>1  0</td>
</tr>
<tr>
<td>0  0</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>1  0</td>
</tr>
<tr>
<td>1  1</td>
</tr>
<tr>
<td>0  1</td>
</tr>
<tr>
<td>^ (exclusive or)</td>
</tr>
<tr>
<td>1  0</td>
</tr>
<tr>
<td>0  1</td>
</tr>
<tr>
<td>0  0</td>
</tr>
<tr>
<td>~ (not)</td>
</tr>
<tr>
<td>1  FALSE</td>
</tr>
<tr>
<td>0  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>00001010</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>11101011</td>
</tr>
<tr>
<td></td>
<td>01001011</td>
<td>11101011</td>
<td></td>
</tr>
<tr>
<td>(A ^ B)</td>
<td>10101010</td>
<td>11100001</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>11100001</td>
<td></td>
</tr>
<tr>
<td>(~A)</td>
<td>10101010</td>
<td>01010101</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

The string operator + can be used 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
```

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></td>
<td>Not equal to</td>
</tr>
<tr>
<td></td>
<td>Transact-SQL extension</td>
</tr>
<tr>
<td></td>
<td>Not greater than</td>
</tr>
<tr>
<td></td>
<td>Transact-SQL extension</td>
</tr>
<tr>
<td></td>
<td>Not less than</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_helpsort` 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:

where column1 between x and y

The following range is not inclusive:

where column1 > x and column1 < y

Using nulls in expressions

Use is null or is not null in queries on columns defined to allow null values.
Expressions

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 (“ ” or ‘ ’) 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 255 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.
Adaptive Server identifiers can be a maximum of 30 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.

**Note** Temporary table names, which begin with the pound sign (#), and local 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.”

### 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. Long temporary table names are truncated to 13 characters (including the pound sign); short names are padded to 13 characters with underscores (_). A 17-digit numeric suffix that is unique for an Adaptive Server session is appended.

### 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 28 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:

set quoted_identifier on

Each time you use the delimited identifier in a statement, you must enclose it in double quotes. For example:

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 coll of 1table, use:

insert "lone"(coll) values ('abc')

Do not not use:
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 coll use:

insert "lone"(coll) values('a''b')

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:

```
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:

```
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.
**Identifiers**

**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

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

**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 271.

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"]
Pattern matching with wildcard characters

An *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%”:

```sql
SELECT phone
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.

```sql
SELECT phone
FROM authors
WHERE phone NOT LIKE '415%'
```

```sql
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”:

```sql
SELECT name
FROM sysobjects
WHERE name LIKE 'sys%'
```

To see all the objects that are *not* system tables, use:

```sql
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 269). 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:

```sql
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:
Pattern matching with wildcard characters

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

To find names that have the characters “en” in them (Bennet, Green, McBadden):

```sql
select au_lname
from authors
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):

```sql
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:

```sql
select au_lname
from authors
where au_lname like "[M-Z]inger"
```

To find both “DeFrance” and “deFrance”:

```sql
select au_lname
from authors
where au_lname like "[dD]eFrance"
```
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”:

```sql
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;]&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;[^a-cdf]&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 (\_, \%, [], ]), 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>
<thead>
<tr>
<th>like predicate</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>like &quot;5@%&quot; escape &quot;@&quot;</td>
<td>5%</td>
</tr>
<tr>
<td>like &quot;_n&quot; escape &quot;_&quot;</td>
<td>_n</td>
</tr>
<tr>
<td>like &quot;%80@%&quot; escape &quot;@&quot;</td>
<td>String containing 80%</td>
</tr>
<tr>
<td>like &quot;_sql_%&quot; escape &quot;_&quot;</td>
<td>String containing _sql%</td>
</tr>
<tr>
<td>like &quot;%#####_##%&quot; escape &quot;#&quot;</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:

```sql
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:

```sql
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>273</td>
</tr>
<tr>
<td>ANSI SQL reserved words</td>
<td>274</td>
</tr>
<tr>
<td>Potential ANSI SQL reserved words</td>
<td>275</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>
<thead>
<tr>
<th>Words</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A</strong></td>
</tr>
<tr>
<td>add, all, alter, and, any, arith_overflow, as, asc, at, authorization, avg</td>
</tr>
<tr>
<td><strong>B</strong></td>
</tr>
<tr>
<td>begin, between, break, browse, bulk, by</td>
</tr>
<tr>
<td><strong>C</strong></td>
</tr>
<tr>
<td>cascade, case, char_convert, check, checkpoint, close, clustered, coalesce, commit, compute, confirm, connect, constraint, continue, controlrow, convert, count, create, current, cursor</td>
</tr>
<tr>
<td><strong>D</strong></td>
</tr>
<tr>
<td>database, dbcc, deallocate, declare, default, delete, desc, deterministic, disk distinct, double, drop, dummy, dump</td>
</tr>
<tr>
<td><strong>E</strong></td>
</tr>
<tr>
<td>else, end, endtran, errlvl, errordata, errorexit, escape, except, exclusive, exec, execute, exists, exit, exp_row_size, external</td>
</tr>
<tr>
<td><strong>F</strong></td>
</tr>
<tr>
<td>fetch, fillfactor, for, foreign, from, func</td>
</tr>
<tr>
<td><strong>G</strong></td>
</tr>
<tr>
<td>goto, grant, group</td>
</tr>
<tr>
<td><strong>H</strong></td>
</tr>
<tr>
<td>having, holdlock</td>
</tr>
</tbody>
</table>

Table 5-1: List of Transact-SQL reserved words
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>
<tr>
<td>Note</td>
</tr>
<tr>
<td>O</td>
</tr>
<tr>
<td>P</td>
</tr>
<tr>
<td>Q</td>
</tr>
<tr>
<td>R</td>
</tr>
<tr>
<td>S</td>
</tr>
<tr>
<td>T</td>
</tr>
<tr>
<td>U</td>
</tr>
<tr>
<td>V</td>
</tr>
<tr>
<td>W</td>
</tr>
</tbody>
</table>

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>
<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>
<tr>
<td>I</td>
</tr>
<tr>
<td>J</td>
</tr>
<tr>
<td>L</td>
</tr>
<tr>
<td>M</td>
</tr>
<tr>
<td>N</td>
</tr>
<tr>
<td>O</td>
</tr>
<tr>
<td>P</td>
</tr>
<tr>
<td>R</td>
</tr>
<tr>
<td>S</td>
</tr>
<tr>
<td>T</td>
</tr>
<tr>
<td>U</td>
</tr>
<tr>
<td>V</td>
</tr>
<tr>
<td>W</td>
</tr>
<tr>
<td>Z</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></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>
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>277</td>
</tr>
<tr>
<td>Exceptions</td>
<td>278</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 only one SQLSTATE warning condition, which is described in Table 6-1:
Exceptions

Adaptive Server Enterprise

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. There are fewer bytes than necessary to form a valid character. | 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. |
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 <code>object_name</code> with unique index <code>index_name</code>.</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 = <code>database_name</code>, table name = <code>table_name</code>, constraint name = <code>constraint_name</code>.</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 = <code>database_name</code>, table name = <code>table_name</code>, constraint name = <code>constraint_name</code>.</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 = <code>database_name</code>, table name = <code>table_name</code>, constraint name = <code>constraint_name</code>.</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 <code>cursor_name</code> 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>
CHAPTER 6  SQLSTATE Codes and Messages

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

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>
Index

Symbols

& (ampersand) “and” bitwise operator 252
* (asterisk)
  for overlength numbers 209
  multiplication operator 251
\ (backslash) character string continuation with 259
::= (BNF notation)
  in SQL statements xvii
^ (caret)
  “exclusive or” bitwise operator 252
  wildcard character 267, 269
: (colon) preceding milliseconds 67, 120
, (comma)
  in default print format for money values 17
  not allowed in money values 18
  in SQL statements xvii
{} (curly braces)
  in SQL statements xvii
$ (dollar sign)
  in identifiers 260
  in money datatypes 18
.. (dots) in database object names 263
= (equals sign) comparison operator 254
> (greater than) comparison operator 254
>= (greater than or equal to) comparison operator 254
< (less than) comparison operator 254
<= (less than or equal to) comparison operator 254
- (minus sign)
  arithmetic operator 251
  for negative monetary values 18
  in integer data 12
!= (not equal to) comparison operator 254
<> (not equal to) comparison operator 254
!= (not greater than) comparison operator 254
!<= (not less than) comparison operator 254
() (parentheses)
  in expressions 258
  in SQL statements xvi
% (percent sign)
  arithmetic operator (modulo) 251
  wildcard character 267
. (period)
  preceding milliseconds 67, 120
  separator for qualifier names 262
| (pipe) “or” bitwise operator 252
+ (plus)
  arithmetic operator 251
  in integer data 12
  null values and 254
  string concatenation operator 253
£ (pound sterling sign)
  in identifiers 260
  in money datatypes 18
“ ” (quotation marks)
  comparison operators and 254
  enclosing constant values 70
  enclosing datetime values 21
  enclosing empty strings 257, 259
  in expressions 259
  literal specification of 259
/ (slash) arithmetic operator (division) 251
[ ] (square brackets)
  character set wildcard 267, 268
  in SQL statements xvii
[^] (square brackets and caret) character set wildcard 267
~ (tilde) “not” bitwise operator 252
_ (underscore)
  object identifier prefix 239, 260
  in temporary table names 260
  character string wildcard 267, 268
¥ (yen sign)
  in identifiers 260
  in money datatypes 18
Index

**Numerics**

“0x” prefix 31, 32
21st century numbers 21

**A**

abbreviations
chars for characters, patindex 169, 171
date parts 66, 120
abort option, lct_admin function 146
abs mathematical function 74
accent sensitivity, wildcard characters and 267
ACF. See Application Context Facility
acos mathematical function 75
adding
   interval to a date 112
timestamp column 229
   user-defined datatypes 44
addition operator (+) 251
aggregate functions 52–58
   See also row aggregates; individual function names
   avg 80
count 102
difference from row aggregates 56
group by clause and 53, 54
having clause and 52
max 159
min 161
   scalar aggregates 53
sum 217
   vector aggregates 53
aggregate functions and cursors 55
all keyword including subqueries 255
alter table command, adding timestamp column 229
   ampersand (&) “and” bitwise operator 252
and (&) bitwise operator 252
   and keyword
   in expressions 257
   range-end 255
angles, mathematical functions for 75
any keyword in expressions 255
   application attributes 194
   Application Context Facility (ACF) 194
   application contexts
   getting 134

listing 153
removing 185
setting 194
approximate numeric datatypes 15
arithabort option, set
   arith_overflow and 10, 63
   mathematical functions and arith_overflow 69
   mathematical functions and numeric_truncation 64, 69
arithignore option, set
   arith_overflow and 63
   mathematical functions and arith_overflow 69
arithmetic
   errors 68
   expressions 250
   operations, approximate numeric datatypes and 15
   operations, exact numeric datatypes and 12
   operations, money datatypes and 17
   operators, in expressions 251
ASCII characters 76
ascii string function 76
asin mathematical function 77
atan mathematical function 78
@@bootcount global variable 243
@@boottime global variable 243
@@bulkarraysize global variable 243
@@bulkbatchsize global variable 243
@@char_convert global variable 243
@@cis_rpc_handling global variable 243
@@cis_version global variable 244
@@client_csexpansion global variable 244
@@client_csname global variable 244
@@client_csname globalization variable 244
@@cmpstate global variable 244
@@connections global variable 244
@@cpu_busy global variable 244
@@datefirst global variable 244
@@dbts global variable 244
@@error global variable 244
@@errorlog global variable 244
@@failedoverconn global variable 244
@@guestuserid global variable 244

Adaptive Server Enterprise
Index

@@hacmpservername global variable 244
@@haconnection global variable 244
@@heapmemsize global variable 244
@@identity global variable 244
@@idle global variable 244
@@invaliduserid global variable 244
@@io_busy global variable 244
@@isolation global variable 244
@@kernel_addr global variable 245
@@kernel_size global variable 245
@@langid global variable 245
@@language global variable 245
@@lock_timeout global variable 245
@@max_connections global variable 245
@@max_precision global variable 245
@@maxcharlen global variable 245
@@maxpagesize global variable 245
@@maxspid global variable 245
@@maxsuid global variable 245
@@maxuserid global variable 245
@@mempool_addr global variable 245
@@min_poolsize global variable 245
@@mingroupid global variable 245
@@minspid global variable 245
@@minsuid global variable 245
@@minuserid global variable 245
@@ncharsize global variable 245
@@nestlevel global variable 245
@@nodeid global variable 246
@@options global variable 246
@@pack_received global variable 246
@@pack_sent global variable 246
@@packet_errors global variable 246
@@pagesize global variable 246
@@parallel_degree global variable 246
@@probesuid global variable 246
@@procid global variable 246
@@recovery_state global variable 246
@@rowcount global variable 246
@@scan_parallel_degree global variable 246
@@servername global variable 246
@@shmem_flags global variable 246
@@spid global variable 246
@@sqlstatus global variable 246
@@stringsize global variable 247
@@tempdbid global variable 247
@@textcolid global variable 43, 247
@@textdbid global variable 43, 247
@@textobjid global variable 43, 247
@@textptr global variable 42, 247
@@textptr_parameters global variable 247
@@textsize global variable 43, 247
@@textts global variable 43, 247
@@thresh_hysteresis global variable 247
@@timeticks global variable 247
@@total_errors global variable 247
@@total_read global variable 247
@@total_write global variable 247
@@tranchained global variable 247
@@trancount global variable 247
@@transactional_rpc global variable 247
@@transstate global variable 247
@@unicharsize global variable 247
@@version global variable 248
@@version_as_integer global variable 248

atn2 mathematical function 79
attributes, setting in an application 194
automatic operations, updating columns with timestamp 18

avg aggregate function 80

B
backslash (\) for character string continuation 259
Backus Naur Form (BNF) notation xvi, xvii
base 10 logarithm function 156
base date 21
between keyword 255

binary

datatypes 31–33
datatypes, “0x” prefix 31, 32
datatypes, trailing zeros in 32
expressions 249
expressions, concatenating 253
representation of data for bitwise operations 252
sort 94, 203

binary datatype 31–33
bit datatype 33
bitwise operators 252–253
blanks
See also spaces, character
character datatypes and 27–30
comparisons 254
empty string evaluated as 259
like and 268
removing leading, with ltrim function 158
removing trailing, with rtrim function 193
BNF notation in SQL statements xvi, xvii
boolean (logical) expressions 249
@@bootcount global variable 243
@@boottime global variable 243
brackets. See square brackets [ ]
browse mode and timestamp datatype 18, 228
built-in function, ACF 194
built-in functions 47–240
See also individual function names
aggregate 52
collection 58
date 66
image 73
mathematical 67
security 69
string 70
system 71
text 73
type conversion 95–99
@@bulkarraysize global variable 243
@@bulkbatchsize global variable 243
by row aggregate subgroup 56

C
calendayofweek date part 120
calendatofyear date part 120
calyearofweek date part 120
case sensitivity
  comparison expressions and 254, 267
  identifiers and 260
  in SQL xviii
cdw. See calendayofweek date part
ceiling mathematical function 82
chains of pages, text or image data 36
char datatype 25–27
  in expressions 258

char string function 84
@@char_convert global variable 243
char_length string function 87
character data, avoiding “NULL” in 257
character datatypes 25–30
character expressions
  blanks or spaces in 27–30
  defined 249
  syntax 250
character sets
  conversion errors 265
  iso_1 265
  multibyte 265
  object identifiers and 265
character strings
  continuation with backslash (\) 259
  empty 259
  specifying quotes within 259
  wildcard in 265
characters
See also spaces, character
“0x” 31, 32
0x 64
deleting, using stuff function 214
number of 87
wildcard 265–271
charindex string function 86
@@cis_rpc_handling global variable 243
@@cis_version global variable 244
client, host computer name and 139
@@client_csexpansion global variable 244
@@client_csid global variable 244
@@client_csname global variable 244
@@cmpstate global variable 244
codes, soundex 205
col_length system function 89
col_name system function 90
colon (:), preceding milliseconds 120
column identifiers. See identifiers.
column name
  as qualifier 262
  in parentheses 56
  returning 90
columns
  identifying 262
  length definition 89
length of numeric, and row aggregates 56
sizes of (list) 2–4
comma (,) default print format for money values 17
not allowed in money values 18
in SQL statements xvii
compare system function 91
comparing values
difference string function 130
in expressions 254
timestamp 228
comparison operators
See also relational expressions in expressions 254
symbols for 254
compute clause and row aggregates 55
computing dates 115
correlation
null values 254
using + operator 253
@@connections global variable 244
constants
and string functions 70
comparing in expressions 258
expression for 249
string functions and 70
continuation lines, character string 259
conventions
See also syntax
identifier name 262
Transact-SQL syntax xvi
used in the Reference Manual xvi
conversion
automatic values 9
between character sets 265
character value to ASCII code 76
dates used with like keyword 24
degrees to radians 178
implicit 9, 258
integer value to character value 84, 227
lower to higher datatypes 258
lowercase to uppercase 230, 231, 232, 233
null values and automatic 9
radians to degrees 127
string concatenation 253
styles for dates 96
uppercase to lowercase 157
convert datatype conversion function 95
concatenation and 253
date styles 96
converting hexadecimal numbers 64
cos mathematical function 100
cot mathematical function 101
count aggregate function 102
CP 850 Alternative
lower case first 94, 203
no accent 94, 203
no case preference 94, 203
CP 850 Scandinavian
dictionary 94, 203
no case preference 94, 203
@@cpu_busy global variable 244
create table command and null values 257
@@curloid global variable 244
curly braces ({} in SQL statements xvii
currency symbols 18, 260
current user
roles of 196
suser_id system function 219
suser_name system function 220
user_id system function 237
user_name system function 238
current_date date function 104
current_timedate function 105
cursors and aggregate functions 55
curatorreservedpgs system function 106
cwk. See calweekofyear date part
cyr. See calyearofweek date part
cyrillic characters 265

D
data_pgs system function 108
database object owners and identifiers 263
database objects
See also individual object names
ID number 167
identifier names 259
user-defined datatypes as 44
database owners
Index

name as qualifier 262, 263
objects and identifiers 263
databases
See also database objects
getting name of 126
ID number, db_id function 125
datalength system function 110
compared to col_length 89
datatype conversions
binary and numeric data 65
bit information 65
character information 60, 61
convert function 95, 98
date and time information 62
domain errors 64, 98
functions for 58–66
hexadecimal-like information 64
hextoint 137
hextoint function 137
image 65, 98
implicit 58
inttohex 143
money information 61
numeric information 61, 62
overflow errors 63
rounding during 61
scale errors 64
datatype precedence. See precedence
datatypes 1–45
See also user-defined datatypes; individual datatype names
approximate numeric 15
binary 31–33
bit 33
date and time 19–25
datetime values comparison 254
decimal 13–14
dropping user-defined 45
exact numeric 11–14
hierarchy 7
integer 12–13
list of 2
mixed, arithmetic operations on 251
synonyms for 2
trailing zeros in binary 32
varbinary 201
date and time datatype 21–25
date datatype 20, 21
date functions 66–67
See also individual function names
current_date 104
current_time 105
dateadd 111
datediff 114
datename 117
datepart 119
day 124
dateadd 136
month 162
year 241
date parts
abbreviation names and values 66, 120
caldayofweek 120
calweekofyear 120
calyearofweek 120
entering 21
order of 22
dateadd date function 111
datediff date function 114
datediff function 115
datefirst option, set 117, 122
dateformat option, set 22
datename date function 117
datepart date function 119
dates
comparing 254
datatypes 19–25
default display settings 23
display formats 20
earliest allowed 21, 66, 112
entry formats 19, 22
pre-1753 datatypes for 66, 112
datetime datatype 21–25
comparison of 254
conversion 24
date functions and values and comparisons 24
day date function 124
day date part 67, 120
dayofyear date part abbreviation and values 67, 120
db_id system function 125, 126
db_name system function 126
DB-Library programs, overflow errors in 81, 218
@@dbts global variable 244

dd. See day date part.

decimal datatype 13–14
decimal numbers
  round function and 189
  str function, representation of 209
decimal points
datatypes, allowing in 13
  in integer data 12
default settings
date display format 20, 23
week day order 122
default Unicode multilingual 94, 203
default values
datatype length 95
datatype precision 95
datatype scale 95
degrees mathematical function 127
degrees, conversion to radians 178
delete command and text row 42
derived_stat system function 128
devices. See sysdevices table.
difference string function 130
division operator (/) 251
dollar sign ($) 260
  in identifiers 260
  in money datatypes 18
domain rules, mathematical functions errors in 68
dots (...) for omitted name elements 263
double datatype 16
double-precision floating-point values 16
doubling quotes
  in expressions 259
  in character strings 28
dropping
  character with stuff function 214
  leading or trailing blanks 158
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 16
    float datatype 5
  money datatypes 18
embedded spaces. See spaces, character.
empty string ('"') or ('")
  not evaluated as null 257
  as a single space 30, 259
enclosing quotes in expressions 259
equal to. See comparison operators
@@error global variable 244
error handling, domain or range 68
@@errorlog global variable 244
ersors
  arithmetic overflow 63
  convert function 60–64, 98
  divide-by-zero 63
domain 64, 98
  scale 64
  trapping mathematical 68
escape characters 270
escape keyword 270–271
european characters in object identifiers 265
exact numeric datatypes 11–14
arithmetic operations and 12
exists keyword in expressions 255
exp mathematical function 131
explicit null value 257
exponent, datatype (e or E)
  approximate numeric types 16
    float datatype 5
  money types 18
exponential value 131
expressions
  defined 249
  enclosing quotes in 259
  including null values 255
  name and table name qualifying 264
  types of 249

F

@@failedoverconn global variable 244

finding
Index

database ID 125
database name 126
server user ID 219
server user name 220, 221, 228, 234
starting position of an expression 86
user aliases 240
user IDs 237
user names 236, 238
valid identifiers 239
first-of-the-months, number of 115
fixed-length columns
  binary datatypes for 31
  character datatypes for 27
  null values in 9
float datatype 16
floating-point data 249
  str character representation of 209
floor mathematical function 132, 133
formats, date. See dates.
free pages, curunreservedpgs system function 106
front-end applications, browse mode and 228
functions 47
  abs mathematical function 74
  acos mathematical function 75
  aggregate 52
  ascii string function 76
  asin mathematical function 77
  atan mathematical function 78
  atn2 mathematical function 79
  avg aggregate function 80
  ceiling mathematical function 82
  char string function 84
  char_length string function 87
  charindex string function 86
  col_length system function 89
  col_name system function 90
  compare system function 91
  conversion 58
  convert datatype conversion function 95
  cos mathematical function 100
  cot mathematical function 101
  count aggregate function 102
  current_date date function 104
  current_time date function 105
  curunreservedpgs system function 106
  data_pgs system function 108
data_length system function 110
date 66
dateded date function 111
datediff date function 114
datename date function 117
datepart date function 119
day date function 124
db_id system function 125, 126
degrees mathematical function 127
derived_stat system function 128
difference string function 130
exp mathematical function 131
floor mathematical function 132
get_appcontext security function 134
getdate date function 136
hextoint datatype conversion function 137
host_id system function 138
host_name system function 139
image 73
index_col system function 141
index_colorder system function 142
inttohex datatype conversion function 143
is_sec_service_on security function 145
isnull system function 144
lct_admin system function 146
left system function 149
len string function 151
license_enabled system function 152
list_appcontext security function 153
lockscheme system function 154
log mathematical function 155
log10 mathematical function 156
lower string function 157
ltrim string function 158
mathematical 67
max aggregate function 159
min aggregate function 161
month date function 162
mut_excl_roles system function 163
newid system function 164
next_identity system function 166
object_id system function 167
object_name system function 168
pagesize system function 169
patindex string function 171
pl mathematical function 174
Index

power mathematical function 175
proc_role system function 176
ptn_data_pgs system function 177
radians mathematical function 178
rand mathematical function 179
replicate string function 180
reserved_pgs system function 181
reverse string function 182
right string function 183
rm_appcontext security function 185
role_contain system function 186
role_id system function 187
role_name system function 188
round mathematical function 189
rowcnt system function 191
rtrim string function 193
security 69
set_appcontext security function 194
show_role system function 196
show_sec_services security function 197
sign mathematical function 198
sin mathematical function 199
sortkey 201
sortkey system function 200
soundex string function 205
space string function 206
sqrt mathematical function 208
square mathematical function 207
str string function 209
str_replace string function 211
string 70
stuff string function 213
substring string function 215
sum aggregate function 217
user_id system function 219
user_name system function 220
syb_quit system function 221
syb_sendmsg 222
system 71
tan mathematical function 223
tempdb_id system function 224
text 73
textptr text and image function 225
textvalid text and image function 226
to_unichar string function 227
tsequal system function 228
uhighsurr string function 230
ulowsurr string function 231
upper string function 232
uscalar string function 233
used_pgs system function 234
user system function 236
user_id system function 237
user_name system function 238
valid_name system function 239
valid_user system function 240
year date function 241

functions, built-in, type conversion 95–99

G

GB Pinyin 94, 203
get_appcontext security function 134
getdate date function 136
global variables
@@bootcount 243
@@boottime 243
@@bulkarraysize 243
@@bulkbatchsize 243
@@char_convert 243
@@cis_rpc_handling 243
@@cis_version 244
@@client_csexpansion 244
@@client_csid 244
@@client_csid 244
@@cmppstate 244
@@connections 244
@@cpu_busy 244
@@curloid 244
@@dts 244
@@error 244
@@errlog 244
@@failedoverconn 244
@@guestuserid 244
@@hapmservername 244
@@hacconnection 244
@@heapmenseize 244
@@identity 244
@@idle 244
@@invaliduserid 244
@@io_busy 244
Index

@@isolation 245
@@kernel_addr 245
@@kernel_size 245
@@langid 245
@@language 245
@@lock_timeout 245
@@max_connections 245
@@max_precision 245
@@maxcharlen 245
@@maxgroupid 245
@@maxpagesize 245
@@maxspid 245
@@maxsuid 245
@@maxuserid 245
@@mempool_addr 245
@@min_poolsize 245
@@mingroupid 245
@@minspid 245
@@minsuid 245
@@minuserid 245
@@ncharsize 245
@@nestlevel 245
@@nodeid 246
@@options 246
@@pack_received 246
@@pack_sent 246
@@packet_errors 246
@@pagesize 246
@@parallel_degree 246
@@probesuid 246
@@procid 246
@@recovery_state 246
@@rowcount 246
@@scan_parallel_degree 246
@@servername 246
@@shmemp_flags 246
@@spid 246
@@sqlstatus 246
@@stringsize 247
@@tempdbid 247
@@textcolid 247
@@textdbid 247
@@textobjid 247
@@textptr 247
@@textptr_parameters 247
@@textsize 247
@@textts 247
@@thresh_hysteresis 247
@@timeticks 247
@@total_errors 247
@@total_read 247
@@total_write 247
@@tranchained 247
@@trancount 247
@@transactional_rpc 247
@@transtate 247
@@unicharsize 247
@@version 248
@@version_as_integer 248
@@datefirst 244
greater than. See comparison operators.
Greek characters 265
group by clause and aggregate functions 53, 54
guest users 237
@@guestuserid global variable 244

H
@@hacmpservername global variable 244
@@haconnection global variable 244
having clause and aggregate functions 52
@@heapmemsize global variable 244
hexadecimal numbers, converting 64
hextoint datatype conversion function 137
hextoint function 137
hh. See hour date part.
hierarchy
    See also precedence
        operators 251
historic dates, pre-1753 66, 112
host computer name 139
host process ID, client process 138
host_id system function 138
host_name system function 139
hour date part 67, 120

I
identifiers 259–265
    case sensitivity and 260
Index

renaming 264
system functions and 239
identities
sa_role and Database Owner 237
server user (suser_id) 220
user (user_id) 237
@@identity global variable 244
identity_burn_max function 140
@@idle global variable 244
IDs, server role and role_id 187
IDs, user
database (db_id) 125
server user 220
user_id function for 219
image datatype 35–44
initializing 40
null values in 40
prohibited actions on 42
image functions 73
implicit conversion of datatypes 9, 258
in keyword in expressions 255
index pages
allocation of 181
system functions 108, 181
total of table and 181
index_col system function 141
index_colorder system function 142
indexes
See also clustered indexes; database objects;
nonclustered indexes
sysindexes table 41
initializing text or image columns 41
inserting
automatic leading zero 32
spaces in text strings 206
int datatype 12
aggregate functions and 81, 218
integer data in SQL 249
integer datatypes, converting to 64
integer remainder. See Modulo operator (%)
internal datatypes of null columns 9
See also datatypes
internal structures, pages used for 109, 181
inttohex datatype conversion function 143
@@invaliduserid global variable 244
@@io_busy global variable 244
is not null keyword in expressions 255
is_sec_service_on security function 145
isnull system function 144
ISO 8859-5 Cyrillic dictionary 94, 204
ISO 8859-5 Russian dictionary 94, 204
ISO 8859-9 Turkish dictionary 94, 204
iso_1 character set 265
@@isolation global variable 245
isql utility command
See also Utility Guide manual
approximate numeric datatypes and 16

J
Japanese character sets and object identifiers 265
joins
count or count(*) with 103
null values and 256

K
@@kernel_addr global variable 245
@@kernel_size global variable 245
keywords 273–276
Transact-SQL 260, 273–274

L
@@langid global variable 245
@@language global variable 245
languages, alternate
effect on date parts 123
weekday order and 122
last-chance threshold and lct_admin function 147
last-chance thresholds 148
latin-1 English, French, German
dictionary 94, 203
no accent 94, 203
no case 94, 203
no case preference 94, 203
latin-1 Spanish
dictionary 94, 203
no accent 94, 204
Index

no case 94, 204

lct_admin system function 146, 148
leading blanks, removal with ltrim function 158
leading zeros, automatic insertion of 32
left system function 149
len string function 151
length
  See also size
  of expressions in bytes 110
  of columns 89
less than. See comparison operators
license_enabled system function 152
like keyword
  searching for dates with 24
  wildcard characters used with 267
linkage, page. See pages, data
list_appcontext security function 153
listing datatypes with types 7
lists
  datatypes 2
  functions 48–52
literal character specification
  like match string 269
  quotes (" ") 259
literal values
  datatypes of 6
  null 257
@@lock_timeout global variable 245
lockscheme system function 154
log mathematical function 154, 155
log10 mathematical function 156
logarithm, base 10 156
logical expressions 249
  syntax 250
  truth tables for 257
log10 mathematical function 156
lower and higher datatypes. See precedence.
lower string function 157
lowercase letters, sort order and 260
  See also case sensitivity
ltrim string function 158

M
macintosh character set 265

matching
  See also Pattern matching
  name and table name 264
mathematical functions 67–69
  abs 74
  acos 75
  asin 77
  atan 78
  atn2 79
  ceiling 82
  cos 100
  cot 101
  degrees 127
  exp 131
  floor 132
  log 155
  log10 156
  pi 174
  power 175
  radians 178
  rand 179
  round 189
  sign 198
  sin 199
  sqrt 208
  square 207
  tan 223
max aggregate function 159
@@max_connections global variable 245
@@max_precision global variable 245
@@maxcharlen global variable 245
@@maxgroupid global variable 245
@@maxpagesize global variable 245
@@maxspid global variable 245
@@maxsuid global variable 245
@@maxuserid global variable 245
@@mempool_addr global variable 245
mi. See minute date part
ml. See minute date part.
  midnights, number of 115
millisecond date part 67, 120
millisecond values, datediff results in 115
min aggregate function 161
@@min_poolsize global variable 245
@@mingroupid global variable 245

296  Adaptive Server Enterprise
Index

@@minspid global variable 245
@@minuid global variable 245
minus sign (-)
in integer data 12
subtraction operator 251
@@minuserid global variable 245
minute date part 67, 120
mixed datatypes, arithmetic operations on 251
mm. See month date part
mm. See month date part.
model database, user-defined datatypes in 44
modulo operator (%) 251
money
default comma placement 17
symbols 260
money datatype 18
arithmetic operations and 17
month date function 162
month date part 67, 120
month values and date part abbreviation 67, 120
ms. See millisecond date part
mutibyte character sets
converting 61
identifier names 265
nchar datatype for 25
wildcard characters and 269
multilingual, Unicode 94, 203
multiplication operator (*) 251
mut_excl_roles system function 163
mutual exclusivity of roles and mut_excl_roles 163

N
“N/A”, using “NULL” or 257
names
See also identifiers
checking with valid_name 264
date parts 66, 120
db_name function 126
finding similar-sounding 205
host computer 139
index_col and index 141
object_name function 168
omitted elements of (...) 263
qualifying database objects 262, 265
suser_name function 220
user_name function 238
weekday function 122
naming
conventions 259–265
database objects 259–265
identifiers 259–265
user-defined datatypes 44
national character. See nchar datatype
natural logarithm 154, 155
nchar datatype 27
@@ncharsize global variable 245
negative sign (-) in money values 18
nesting
aggregate functions 53
string functions 70
@@nestlevel global variable 245
newid system function 164
next_identity system function 166
@@nodeid global variable 246
“none”, using “NULL” or 257
not keyword in expressions 255
not like keyword 266
not null values
spaces in 30
not null values in spaces 30
null keyword in expressions 255
null string in character columns 214, 257
null values
column datatype conversion for 30
default parameters as 256
in expressions 256
text and image columns 40
null values in a where clause 256
number (quantity of)
first-of-the-months 115
midnights 115
rows in count(*) 102
rows reported by rowcnt 191
Sundays 115
number of characters and date interpretation 24
number of pages
allocated to table or index 181
reserved_pgs function 181
used by table and clustered index (total) 234
used by table or index 108

Reference Manual: Building Blocks 297
used_pgs function 234

numbers
  asterisks (**) for overlength 209
  converting strings of 30
  database ID 125
  object ID 167
  odd or even binary 32
  random float 179
  weekday names and 122

numeric data and row aggregates 56
numeric datatype 13
  range and storage size 3
numeric expressions 249
  round function for 189
  nvarchar datatype 27
  spaces in 27

O

object Allocation Map (OAM) pages 235
object names, database
  See also identifiers
    user-defined datatype names as 44
object_id system function 167
object_name system function 168
objects. See database objects; databases
  operators
    arithmetic 251
    bitwise 252–253
    comparison 254
    precedence 251
  @@options global variable 246
  or keyword in expressions 257
  order
    See also indexes; precedence; sort order
    of execution of operators in expressions 251
    of date parts 22
    reversing character expression 182
    weekday numeric 122
  order by clause 201
other users, qualifying objects owned by 265
overflow errors in DB-Library 81, 218
ownership of objects being referenced 265

P

@@pack_received global variable 246
@@pack_sent global variable 246
@@packet_errors global variable 246
padding, data
  blanks and 27
  underscores in temporary table names 260
  with zeros 32
pages, data
  allocation of 181
  chain of 36
  data_pgs system function 108
  reserved_pgs system function 181
  used for internal structures 109, 181
  used in a table or index 108, 234
  used_pgs system function 234
  pages, index
    number used in nonclustered 234
  pages, OAM (Object Allocation Map), number of 235
  @@pagesize global variable 246
  pagesize system function 169
  @@parallel_degree global variable 246
  parentheses ()
    See also Symbols section of this index
    in an expression 258
    in SQL statements xvi
  partitioned tables, size of 177
  patindex string function 172
    text/image function 43
  pattern matching 265
    See also String functions; wildcard characters
    charindex string function 86
    difference string function 130
    patindex string function 172
  percent sign (%)
    modulo operator 251
    wildcard character 267
  period ()
    preceding milliseconds 120
    separator for qualifier names 262
  pi mathematical function 174
  platform-independent conversion
    hexadecimal strings to integer values 137
    integer values to hexadecimal strings 143
  plus (+)
    arithmetic operator 251
Index

in integer data 12
null values and 254
string concatenation operator 253

pointers
null for uninitialized text or image column 225
text and image page 225
text or image column 40, 43
pound sterling sign (£)
in identifiers 260
in money datatypes 18

power mathematical function 175
precedence
of lower and higher datatypes 258
of operators in expressions 251
preceding blanks. See blanks; spaces, character precision, datatype
approximate numeric types 16
exact numeric types 13
money types 17
@@probesuid global variable 246
proc_role system function 176
@@procid global variable 246
ptn_data_pgs system function 177
punctuation, characters allowed in identifiers 260

Q
qq. See quarter date part
qualifier names 262, 265
quarter date part 66, 120
quotation marks (“ ”)
comparison operators and 254
for empty strings 257, 259
enclosing constant values 70
in expressions 259
literal specification of 259

R
radians mathematical function 178
radians, conversion to degrees 127
rand mathematical function 179
range
See also numbers; size

of date part values 66, 120
datediff results 115
errors in mathematical functions 68
money values allowed 17
of recognized dates 21
wildcard character specification of 268, 269
range queries
and end keyword 255
between start keyword 255
readtext command and text data initialization requirement 41
real datatype 16
@@recovery state global variable 246
reference information
datatypes 1
reserved words 273
Transact-SQL functions 47
relational expressions 250
See also comparison operators
removing application contexts 185
replicate string function 180
reserve option, lct_admin function 146
reserved words 273–276
See also keywords
database object identifiers and 259, 260
SQL92 274
Transact-SQL 273–274
reserved_pgs system function 181
results of row aggregate operations 55
retrieving similar-sounding words or names 205
reverse string function 182
right string function 183, 184
right-justification of str function 210
rm_appcontext security function 185
role hierarchies and role_contain 186
role_contain system function 186
role_id system function 187
role_name system function 188
roles
checking with proc_role 176
showing system with show_role 196
roles, user-defined and mutual exclusivity 163
round mathematical function 189
rounding 189
approximate numeric datatypes 16
datetime values 20, 62
Index

money values 17, 61
str string function and 209
row aggregates 55
compute and 55
difference from aggregate functions 56
rowcnt system function 191
@@rowcount global variable 246
rows, table
detail and summary results 55
number of 191
row aggregates and 55
rtrim string function 193
rules. See database objects.

S
scalar aggregates and nesting vector aggregates within 53
scale, datatype 13
decimal 8
IDENTITY columns 13
loss during datatype conversion 11
numeric 8
@@scan_parallel_degree global variable 246
search conditions and datetime data 24
second date part 67, 120
seconds, datediff results in 115
security functions 69
get_appcontext 134
is_sec_service_on 145
list_appcontext 153
rm_appcontext 185
set_appcontext 194
show_sec_services 197
seed values and rand function 179
select command 201
aggregates and 52
for browse 228
restrictions in standard SQL 54
in Transact-SQL compared to standard SQL 54
select into command not allowed with compute 58
server user name and ID
suser_id function 219
suser_name function for 220
@@servername global variable 246
set_appcontext security function 194
setting application context 194
shift-JIS binary order 94, 204
@@shmem_flags global variable 246
show_role system function 196
show_sec_services security function 197
sign mathematical function 198
similar-sounding words. See soundex string function
sin mathematical function 199
single quotes. See quotation marks
single-byte character sets, char datatype for 25
size
See also length; number (quantity of); range; size
limit; space allocation
column 89
floor mathematical function 133
identifiers (length) 260
image datatype 35
of pi 174
text datatype 35
size limit
approximate numeric datatypes 16
binary datatype 31
char columns 27
datatypes 2–4
datetime datatype 21
double precision datatype 16
exact numeric datatypes 12
fixed-length columns 27
float datatype 16
image datatype 31
integer value smallest or largest 133
money datatypes 18
nchar columns 27
nvarchar columns 27
real datatype 16
smalldatetime datatype 27
varbinary datatype 31
varchar columns 27
slash (/) division operator 251
smalldatetime datatype 21
date functions and 120
smallint datatype 12
smallmoney datatype 18
sort order
character collation behavior 200, 201
comparison operators and 254
<table>
<thead>
<tr>
<th>Function/Procedure/Keyword</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>sortkey function</td>
<td>201</td>
</tr>
<tr>
<td>sortkey system function</td>
<td>200</td>
</tr>
<tr>
<td>soundex string function</td>
<td>205</td>
</tr>
<tr>
<td>sp_bindefault system</td>
<td>45</td>
</tr>
<tr>
<td>sp_bindrule system</td>
<td>45</td>
</tr>
<tr>
<td>sp_help system procedure</td>
<td>45</td>
</tr>
<tr>
<td>space string function</td>
<td>206</td>
</tr>
<tr>
<td>spaces, character</td>
<td></td>
</tr>
<tr>
<td>See also blanks</td>
<td></td>
</tr>
<tr>
<td>in character datatypes</td>
<td>27–30</td>
</tr>
<tr>
<td>empty strings (&quot; &quot;) or (&quot; &quot;) as</td>
<td>257, 259</td>
</tr>
<tr>
<td>inserted in text strings</td>
<td>206</td>
</tr>
<tr>
<td>like datetime values</td>
<td>25</td>
</tr>
<tr>
<td>not allowed in identifiers</td>
<td>260</td>
</tr>
<tr>
<td>speed (Server)</td>
<td></td>
</tr>
<tr>
<td>binary and varbinary</td>
<td>31</td>
</tr>
<tr>
<td>datatype access</td>
<td></td>
</tr>
<tr>
<td>@@spid global variable</td>
<td>246</td>
</tr>
<tr>
<td>SQL (used with Sybase databases). See Transact-SQL</td>
<td></td>
</tr>
<tr>
<td>SQL standards</td>
<td></td>
</tr>
<tr>
<td>aggregate functions and</td>
<td>54</td>
</tr>
<tr>
<td>concatenation and</td>
<td>254</td>
</tr>
<tr>
<td>SQLSTATE codes</td>
<td>277–283</td>
</tr>
<tr>
<td>exceptions</td>
<td>278–283</td>
</tr>
<tr>
<td>@@sqlstatus global variable</td>
<td>246</td>
</tr>
<tr>
<td>sqrt mathematical function</td>
<td>208</td>
</tr>
<tr>
<td>square brackets [ ]</td>
<td></td>
</tr>
<tr>
<td>caret wildcard character [*] and</td>
<td>267, 269</td>
</tr>
<tr>
<td>in SQL statements</td>
<td>xvii</td>
</tr>
<tr>
<td>wildcard specifier</td>
<td>267</td>
</tr>
<tr>
<td>square mathematical function</td>
<td>207</td>
</tr>
<tr>
<td>square root mathematical function</td>
<td>208</td>
</tr>
<tr>
<td>ss. See second date part</td>
<td></td>
</tr>
<tr>
<td>storage management for text and image data</td>
<td>41</td>
</tr>
<tr>
<td>str string function</td>
<td>209</td>
</tr>
<tr>
<td>str_replace string function</td>
<td>211</td>
</tr>
<tr>
<td>string functions</td>
<td>70–71</td>
</tr>
<tr>
<td>See also text datatype</td>
<td></td>
</tr>
<tr>
<td>ascii</td>
<td>76</td>
</tr>
<tr>
<td>char</td>
<td>84</td>
</tr>
<tr>
<td>char_length</td>
<td>87</td>
</tr>
<tr>
<td>charindex</td>
<td>86</td>
</tr>
<tr>
<td>difference</td>
<td>130</td>
</tr>
<tr>
<td>len</td>
<td>151</td>
</tr>
<tr>
<td>lower</td>
<td>157</td>
</tr>
<tr>
<td>ltrim</td>
<td>158</td>
</tr>
<tr>
<td>patindex</td>
<td>171</td>
</tr>
<tr>
<td>replicate</td>
<td>180</td>
</tr>
<tr>
<td>reverse</td>
<td>182</td>
</tr>
<tr>
<td>right</td>
<td>183</td>
</tr>
<tr>
<td>rtrim</td>
<td>193</td>
</tr>
<tr>
<td>soundex</td>
<td>205</td>
</tr>
<tr>
<td>space</td>
<td>206</td>
</tr>
<tr>
<td>str</td>
<td>209</td>
</tr>
<tr>
<td>stuff</td>
<td>213</td>
</tr>
<tr>
<td>substring</td>
<td>215</td>
</tr>
<tr>
<td>to_unichar</td>
<td>227</td>
</tr>
<tr>
<td>uhighsurr</td>
<td>230</td>
</tr>
<tr>
<td>ulowsurr</td>
<td>231</td>
</tr>
<tr>
<td>upper</td>
<td>232</td>
</tr>
<tr>
<td>uscalar</td>
<td>233</td>
</tr>
<tr>
<td>strings, concatenating</td>
<td>253</td>
</tr>
<tr>
<td>@@stringsize global variable</td>
<td>247</td>
</tr>
<tr>
<td>stuff string function</td>
<td>213, 214</td>
</tr>
<tr>
<td>style values, date</td>
<td>96</td>
</tr>
<tr>
<td>representation</td>
<td></td>
</tr>
<tr>
<td>subqueries</td>
<td></td>
</tr>
<tr>
<td>any keyword and</td>
<td>255</td>
</tr>
<tr>
<td>in expressions</td>
<td>255</td>
</tr>
<tr>
<td>substring</td>
<td>215</td>
</tr>
<tr>
<td>subtraction operator (-)</td>
<td>251</td>
</tr>
<tr>
<td>sum aggregate function</td>
<td>217</td>
</tr>
<tr>
<td>sundays, number value</td>
<td>115</td>
</tr>
<tr>
<td>suser_id system function</td>
<td>219</td>
</tr>
<tr>
<td>suser_name system function</td>
<td>220</td>
</tr>
<tr>
<td>syb_quit system function</td>
<td>221</td>
</tr>
<tr>
<td>syb_sendmsg function</td>
<td>222</td>
</tr>
<tr>
<td>symbols</td>
<td></td>
</tr>
<tr>
<td>See also wildcard characters; Symbols section of this index</td>
<td></td>
</tr>
<tr>
<td>arithmetic operator</td>
<td>251</td>
</tr>
<tr>
<td>comparison operator</td>
<td>254</td>
</tr>
<tr>
<td>in identifier names</td>
<td>260</td>
</tr>
<tr>
<td>matching character strings</td>
<td>267</td>
</tr>
<tr>
<td>money</td>
<td>260</td>
</tr>
<tr>
<td>in SQL statements</td>
<td>xvi</td>
</tr>
<tr>
<td>wildcards</td>
<td>267</td>
</tr>
<tr>
<td>synonyms and chars and characters, patindex</td>
<td>171</td>
</tr>
<tr>
<td>synonyms for datatypes</td>
<td>2</td>
</tr>
<tr>
<td>synonyms, chars and characters, patindex</td>
<td>169</td>
</tr>
<tr>
<td>syntax conventions, Transact-SQL</td>
<td>xvi</td>
</tr>
</tbody>
</table>

Reference Manual: Building Blocks 301
## Index

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>syscolumns</td>
<td>33</td>
</tr>
<tr>
<td>sysindexes</td>
<td>41</td>
</tr>
<tr>
<td>syssrvroles</td>
<td>187</td>
</tr>
<tr>
<td>system datatypes</td>
<td>71–72</td>
</tr>
<tr>
<td>col_length</td>
<td>89</td>
</tr>
<tr>
<td>col_name</td>
<td>90</td>
</tr>
<tr>
<td>compare</td>
<td>91</td>
</tr>
<tr>
<td>curunreservedpgs</td>
<td>106</td>
</tr>
<tr>
<td>data_pgs</td>
<td>108</td>
</tr>
<tr>
<td>datalength</td>
<td>110</td>
</tr>
<tr>
<td>db_id</td>
<td>125, 126</td>
</tr>
<tr>
<td>derived_stat</td>
<td>128</td>
</tr>
<tr>
<td>host_id</td>
<td>138</td>
</tr>
<tr>
<td>host_name</td>
<td>139</td>
</tr>
<tr>
<td>index_col</td>
<td>141</td>
</tr>
<tr>
<td>index_colorder</td>
<td>142</td>
</tr>
<tr>
<td>isnull</td>
<td>144</td>
</tr>
<tr>
<td>lct_admin</td>
<td>146</td>
</tr>
<tr>
<td>left</td>
<td>149</td>
</tr>
<tr>
<td>license_enabled</td>
<td>152</td>
</tr>
<tr>
<td>lockscheme</td>
<td>154</td>
</tr>
<tr>
<td>mut_excl_roles</td>
<td>163</td>
</tr>
<tr>
<td>newid</td>
<td>164</td>
</tr>
<tr>
<td>next_identity</td>
<td>166</td>
</tr>
<tr>
<td>object_id</td>
<td>167</td>
</tr>
<tr>
<td>object_name</td>
<td>168</td>
</tr>
<tr>
<td>pagesize</td>
<td>169</td>
</tr>
<tr>
<td>proc_role</td>
<td>176</td>
</tr>
<tr>
<td>ptn_data_pgs</td>
<td>177</td>
</tr>
<tr>
<td>reserved_pgs</td>
<td>181</td>
</tr>
<tr>
<td>role_contain</td>
<td>186</td>
</tr>
<tr>
<td>role_id</td>
<td>187</td>
</tr>
<tr>
<td>role_name</td>
<td>188</td>
</tr>
<tr>
<td>rowcnt</td>
<td>191</td>
</tr>
<tr>
<td>show_role</td>
<td>196</td>
</tr>
<tr>
<td>sortkey</td>
<td>200</td>
</tr>
<tr>
<td>suser_id</td>
<td>219</td>
</tr>
<tr>
<td>suser_name</td>
<td>220</td>
</tr>
<tr>
<td>syb_quit</td>
<td>221</td>
</tr>
<tr>
<td>tempdb_id</td>
<td>224</td>
</tr>
<tr>
<td>tequal</td>
<td>228</td>
</tr>
<tr>
<td>used_pgs</td>
<td>234</td>
</tr>
<tr>
<td>user</td>
<td>236</td>
</tr>
<tr>
<td>user_id</td>
<td>237</td>
</tr>
<tr>
<td>user_name</td>
<td>238</td>
</tr>
<tr>
<td>valid_name</td>
<td>239</td>
</tr>
<tr>
<td>valid_user</td>
<td>240</td>
</tr>
<tr>
<td>system roles and show_role and</td>
<td>196</td>
</tr>
<tr>
<td>system tables and sysname datatype</td>
<td>34</td>
</tr>
</tbody>
</table>

### T

table pages

See also pages, data system functions 108
tables

identifying 262
names as qualifiers 262
worktables 52
tan mathematical function 223
tangents, mathematical functions for 223
tempdb database, user-defined datatypes in 44
@@tempdbid global variable 247
tempdb_id system function 224
tempdbs and tempdb_id system function 224
temporary tables, naming 260
text and image functions
textptr 225
textvalid 226
text datatype 35–44

calculate command 43
converting 61
initializing with null values 40
null values 40
prohibited actions on 42
text datatype and ascii string function 76
text functions 73
text page pointer 89
text pointer values 225
@@textcolid global variable 43, 247
@@textidbid global variable 43, 247
@@textobjid global variable 43, 247
textptr function 225
@@textptr global variable 42, 247
textptr text and image function 225
@@textptr_parameters global variable 247
@@textsize global variable 43, 247
@@texts global variable 43, 247
textvalid text and image function 226
Thai dictionary 94, 204

302

Adaptive Server Enterprise
Index

@@thresh_hysteresis global variable 247
thresholds, last-chance 148
time datatype 21
time values
datatypes 19–25
timestamp datatype 18–19
automatic update of 18
browse mode and 18, 228
comparison using tsequal function 228
@@timeticks global variable 247
tinyint datatype 12
to_unichar string function 227
@@total_errors global variable 247
@@total_read global variable 247
@@total_write global variable 247
trailing blanks. See blanks
@@tranchained global variable 247
@@trancount global variable 247
@@transactional_rpc global variable 247
Tranact-SQL
aggregate functions in 54
reserved words 273–274
translation of integer arguments into binary numbers 252
@@transtate global variable 247
triggers See database objects; stored procedures.
trigonometric functions 67, 67–223
true/false data, bit columns for 33
truncation
 articabort numeric_truncation 10
binary datatypes 31
character string 27
datediff results 115
str conversion and 210
temporary table names 260
truth tables for logical expressions 257
tsequal system function 228
twenty-first century numbers 21
character string wildcard 267, 268
object identifier prefix 239, 260
in temporary table names 260
@@unicastsize global variable 247
unicode multilingual, default 94, 203
unique names as identifiers 261
updating
See also changing 18
in browse mode 228
prevention during browse mode 228
upper string function 232, 233
uppercase letter preference 260
See also case sensitivity; order by clause
us_english language, weekdays setting 122
uscalar string function 233
used_pgs system function 234
User Datagram Protocol messaging 222
user IDs
user_id function for 237
valid_user function 240
user names 238
user names, finding 220, 238
user objects. See database objects
user system function 236
user_id system function 237
user_name system function 238
user-created objects. See database objects
user-defined datatypes
See also datatypes
creating 44
dropping 45
sysname as 34
user-defined roles and mutual exclusivity 163
using bytes option, patindex string function 169, 171, 172

V
valid_name system function 239
using after changing character sets 264
valid_user system function 240
varbinary datatype 31–33, 201
varchar datatype 27
datetime values conversion to 24
in expressions 258

UDP messaging 222
uhighsurr string function 230
ulowsurr string function 231
underscore (_) 33

Reference Manual: Building Blocks 303
Index

spaces in 27
variable-length character. See varchar datatype 53
vector aggregates 53
nesting inside scalar aggregates 53
@@version global variable 248
@@version_as_integer global variable 248
view name in qualified object name 262

W
week date part 67, 120
weekday date part 67, 120
weekday date value, names and numbers 122
where clause, null values in a 256
wildcard characters 265–271
See also patindex string function
in a like match string 267
literal characters and 269
used as literal characters 269
wk. See week date part
words, finding similar-sounding 205
worktables, number of 52
writetext command and text data initialization requirement 41

Y
year date function 241
year date part 66, 120
yen sign (¥) 260
in identifiers 260
in money datatypes 18
yes/no data, bit columns for 33
yy. See year date part
yy. See year date part.

Z
zero x (0x) 31, 32, 64
zeros, trailing, in binary datatypes 32–33