Example:

SQL

```
EXEC sp_addtype empidtype , 'DEC(4)' , 'NULL'

  CREATE TABLE emps2 (
    empid empidtype  PRIMARY KEY, -- makes it NOT NULL
    ename    VARCHAR(20)
  )

  INSERT INTO emps2 VALUES ( 1111 , 'James Bond' )

EXEC  sp_help   empidtype
```

Result

Type_name	Storage_type	Length	Prec	Scale	Nullable	...
---	---	---	---	---	---	...
empidtype	decimal	5	4	0	yes	...

SQL

```
EXEC  sp_help   emps2
-- Partial listing to show the new data type for empid column
```

Result

Column_name	Type	...
---	---	...
empid	empidtype	...
ename	varchar	...

2.6 TRANSACT-SQL FUNCTIONS

Microsoft SQL Server 2000 has a large number of built-in functions available for SQL programming, database administration and other purposes. The major categories are listed in Table 2-51.

The following sections contain detailed explanations for each of the categories from Table 2-51, categorizing the functions and describing their general pur-

Table 2-51 Built-in Functions

Function Category	Description
Aggregate Functions	Perform a calculation on a set of values and return a single summarizing value, e.g., COUNT(), SUM(), AVG()
Cursor Functions	Returns information about cursors.
Configuration Functions	Returns information about the current configuration.
Date and Time Functions	Performs an operation on a date and time input value and returns either a string, numeric or date and time value.
Mathematical Functions	Performs a calculation based on input values provided as parameters to the function, and returns a numeric value.
Metadata Functions	Returns information about the database and database objects.
Rowset Functions	Returns an object that can be used in an SQL statement like a table.
Security Functions	Returns information about users and roles.
String Functions	Performs an operation on a string (CHAR or VARCHAR) input value and returns a string or numeric value.
System Functions	Performs operations and returns information about values, objects and settings in Microsoft SQL Server.
System Statistical Functions	Returns statistical information about the system.
Text and Image Functions	Performs an operation on a text or image input values or column and returns information about the value.

pose. Some of the commonly used functions are covered in greater detail, with examples. All functions can be found in Books Online.

See also Deterministic and Nondeterministic Functions, page 162.

2.6.1 Aggregate Functions

Aggregate functions perform a calculation on a set of values and return a single summarizing value. Table 2-52 lists the function names and their details.COUNT(), AVG(), MIN(), MAX() and SUM() are ANSI SQL-92 and 99 standard. All are deterministic (see page 162).

Aggregate functions are only allowed as expressions in the following cases.

- The select list of a SELECT statement (either a subquery or an outer query).
- A COMPUTE or COMPUTE BY clause.
- A HAVING clause.

Table 2-52 Aggregate Functions

Aggregate Function Name	Description and Syntax
AVG	Returns the average of the values in a group. Null values are ignored.
BINARY_CHECKSUM	Returns the binary checksum value computed over a row of a table or over a list of expressions. It can be used to detect changes to a row of a table. *Syntax:* `BINARY_CHECKSUM ( * | expression [ ,...n ] )`
CHECKSUM	Returns the checksum value computed over a row of a table, or over a list of expressions. CHECKSUM is intended for use in building hash indices. *Syntax:* `CHECKSUM ( * | expression [ ,...n ] )`
CHECKSUM_AGG	Returns the checksum of the values in a group. Null values are ignored. *Syntax:* `CHECKSUM_AGG ( [ ALL | DISTINCT ] expression )`
COUNT	Returns the number of items in a group as an INT data type value. *Syntax:* `COUNT ( { [ ALL | DISTINCT ] expression ] | * } )`
COUNT_BIG	Returns the number of items in a group as a BIGINT data type value. *Syntax:* `COUNT_BIG ( { [ ALL | DISTINCT ] expression } | * )`
GROUPING	Works only in SELECT statement with GROUP BY plus either ROLLUP or CUBE to determine whether a NULL in the result set was generated by ROLLUP/CUBE or comes from NULL value(s) in the underlying data. *Syntax:* `GROUPING` (*column_name*) returns 1 if a NULL under *column_name* is from ROLLUP or CUBE returns 0 if a NULL under *column_name* is from the data See examples with ROLLUP and CUBE.
MAX	Returns the maximum value in the expression. *Syntax:* `MAX ( [ ALL | DISTINCT ] expression )`
MIN	Returns the minimum value in the expression. *Syntax:* `MIN ( [ ALL | DISTINCT ] expression )`
SUM	Returns the sum of the values in the expression. SUM can be used with numeric columns only. Null values are ignored. *Syntax:* `SUM ( [ ALL | DISTINCT ] expression )`
STDEV	Returns the sample statistical standard deviation of all values in the given expression. For sample statistics the divisor is (n-1). *Syntax:* `STDEV ( expression )  )`
STDEVP	Returns the population statistical standard deviation for all values in the given expression. For population statistics the divisor is (n). *Syntax:* `STDEVP ( expression )  )`

Table 2-52 Aggregate Functions (cont.)

VAR	Returns sample statistical variance of all values in the given expression. For sample statistics the divisor is (n-1). *Syntax:* `VAR ( expression )`
VARP	Returns the population statistical variance for all values in the given expression. For population statistics the divisor is (n). *Syntax:* `VARP ( expression )  )`

2.6.2 Cursor Functions

Cursor Functions, listed in Table 2-53, return cursor status information. All are nondeterministic (see page 162).

Table 2-53 Cursor Functions

Function Name	Description and Syntax
@@CURSOR_ROWS	Returns the number of rows in the last cursor opened on the connection.
@@FETCH_STATUS	Returns the status of the last cursor FETCH statement issued against any cursor currently opened by the connection. Global function to all cursors in the connection, so use it immediately after the FETCH whose status you're interested in.
CURSOR_STATUS	A scalar function that allows the caller of a stored procedure to determine whether or not the procedure has returned a cursor and result set.

These functions are discussed in more detail in Cursors, page 638.

2.6.3 Configuration Functions

Configuration functions, listed in Table 2-54, return information about the current server and database configuration settings. All are nondeterministic (see page 162).

Table 2-54 Configuration Functions

Configuration Function Name	Description (Syntax is just the Function Name since all are read-only and none take parameters.)
@@DATEFIRST	Returns the current value of the SET DATEFIRST parameter, which indicates the specified first day of each week: 1 for Monday, 2 for Wednesday, and so on through 7 for Sunday. The U.S. English default is 7, Sunday.

Table 2-54 Configuration Functions (cont.)

Configuration Function Name	Description (Syntax is just the Function Name since all are read-only and none take parameters.)
	Syntax: **--** Syntax for all functions in this table is just the function name @@DATEFIRST **Example:** **SQL:** `SET DATEFIRST 1 -- Sets session value, See Books Online` `SELECT @@DATEFIRST As'Beginning of Week'` **Result:** Beginning of Week ---------- 1
@@DBTS	Returns the value of the current **timestamp** data type for the current database. This **timestamp** is guaranteed to be unique in the database. *Syntax:* **@@DBTS**
@@LANGID	Returns local language identifier (ID) of the language currently in use.
@@LANGUAGE	Returns the name of the language currently in use.
@@LOCK_TIMEOUT	Returns the current lock time-out setting, in milliseconds, for the current session.
@@MAX_CONNECTIONS	Returns the maximum number of simultaneous user connections allowed on a Microsoft SQL Server. The number returned is not necessarily the number currently configured.
@@MAX_PRECISION	Returns the precision level used by **decimal** and **numeric** data types as currently set in the server.
@@NESTLEVEL	Returns the nesting level of the current stored procedure execution (initially 0).
@@OPTIONS	Returns information about current SET options. See description page 204.
@@REMSERVER	Returns the name of the remote Microsoft SQL Server database server as it appears in the login record. It enables a stored procedure to check the name of the database server from which the procedure is run.
@@SERVERNAME	Returns the name of the local server running Microsoft SQL Server.
@@SERVICENAME	Returns the name of the registry key under which Microsoft SQL Server is running. @@SERVICENAME returns MSSQLServer if the current instance is the default instance; this function returns the instance name if the current instance is a named instance.

Table 2-54 Configuration Functions (cont.)

Configuration Function Name	Description (Syntax is just the Function Name since all are read-only and none take parameters.)
@@SPID	Returns the server process identifier (ID) of the current user process.
@@TEXTSIZE	Returns the current value of the TEXTSIZE option of the SET statement, which specifies the maximum length, in bytes, of **text** or **image** data that a SELECT statement returns.
@@VERSION	Returns the date, version and processor type for the current installation of Microsoft SQL Server.

Example:

```
SQL
SELECT @@SERVERNAME As Server , @@SERVICENAME Service

Result
Server          Service
--------------- ----------------------
AMY             MSSQLServer
```

2.6.4 Date and Time Functions

Date and time functions perform an operation on a date and time input value and return either a string, numeric or date and time value. See Table 2-55.

These functions are deterministic and nondeterministic. See details page 163. DATENAME, GETDATE and GETUTCDATE are nondeterministic. DATEPART is deterministic unless used with dw datepart. The rest are deterministic.

Table 2-55 Date and Time Functions

Date Function Name	Description and Syntax
DATEADD	Returns a new **datetime** value after adding an interval (number argument) to the specified date argument. The interval is an integer whose date/time units are specified by the datepart argument as in DATEDIFF below. *Syntax:* `DATEADD ( datepart , number, date )`

Table 2-55 Date and Time Functions (cont.)

Date Function Name	Description and Syntax
	Example: **SQL:** `SELECT DATEADD(week, 1, '1 Jan, 2002' ) As '2d week in 2002'` **Result:** 2d week in 2002 --- 2002-01-08 00:00:00.000
DATEDIFF	Returns the integer difference between two DATETIME arguments in the date or time increments specified by the datepart argument (year, quarter, ..., minute, ...). *Syntax:* `( datepart , startdate , enddate )` **Example:** **SQL:** `SELECT DATEDIFF (week, '1 Jan, 2002', '19 Jan, 2002' ) As NumWeeks` **Result:** NumWeeks --- 2
DATENAME	Returns a character string representing the specified datepart of the specified date. *Syntax:* `DATENAME ( datepart , date )` **Example:** **SQL:** `SELECT DATENAME (month, '1 Jan, 2002' ) As '1st Month in 2002'` **Result:** 1st Month in 2002 --- January
DATEPART	Returns an integer representing the specified datepart of the specified date. *Syntax:* `DATEPART ( datepart , date )` **Example:** **SQL:** `SELECT DATEPART ( month, '1 Jan, 2002' ) As '1st Month in 2002'` **Result:** 1st Month in 2002 --- 1

Table 2-55 Date and Time Functions (cont.)

Date Function Name	Description and Syntax
DAY	Returns an integer representing the day datepart of the specified date. *Syntax:* `DAY ( date )` **Example:** **SQL:** `SELECT DAY( '1 Jan, 2002' ) As 'Day of Month'` **Result:** Day of Month --- 1 Note: DAY(date) is equivalent to: DATEPART(dd, date)
MONTH	Returns an integer that represents the month part of a specified date. *Syntax:* `MONTH ( date )` **Example:** **SQL:** `SELECT MONTH( '1 Jan, 2002' ) As 'Month Number'` **Result:** Month Number --- 1 Note: MONTH(date) is equivalent to: DATEPART(mm, date)
YEAR	Returns an integer that represents the year part of a specified date. *Syntax:* `YEAR ( date )` **Example:** **SQL:** `SELECT YEAR( '1 Jan, 2002' ) As Year` **Result:** Year --- 2002 Note: YEAR(date) is equivalent to: DATEPART(yy, date)
GETDATE	Returns the current system date and time in the Microsoft SQL Server standard internal format for datetime values. *Syntax:* `GETDATE ( )` **Example:** **SQL:** `SELECT GETDATE() As Today` **Result:** Today --- 2002-03-27 17:26:14.723

Table 2-55 Date and Time Functions (cont.)

Date Function Name	Description and Syntax
GETUTC-DATE	Returns the datetime value representing the current UTC time (Universal Time Coordinate or Greenwich Mean Time). The current UTC time is derived from the current local time and the time zone setting in the operating system of the computer on which SQL Server is running. *Syntax:* `GETUTCDATE()` **Example:** **SQL:** `SELECT GETDATE() As Now , GETUTCDATE() As NowUTC -- From PST` **Result:** Now NowUTC ---------- ---------- 2002-03-27 16:29:13.250 2002-03-27 23:29:13.250

2.6.5 Mathematical Functions

A mathematical function performs a calculation based on input values provided as parameters to the function and returns a numeric value (Table 2-56). All are deterministic except RAND. See listing page 162.

Table 2-56 Mathematical Functions

Math Function Name	Description and Syntax
ABS	Returns the absolute, positive value of the given numeric expression. *Syntax:* `( numeric_expression )`
ACOS	Returns the angle, in radians, (arccosine) whose cosine is the given float expression. *Syntax:* `ACOS ( float_expression )`
ASIN	Returns the angle, in radians, (arcsine) whose sine is the given float expression. *Syntax:* `ASIN ( float_expression )`
ATAN	Returns the angle in radians (arctangent) whose tangent is the given float expression. *Syntax:* `ATAN ( float_expression )`
ATN2	Returns the angle, in radians, whose tangent is between the two given float expressions. *Syntax:* `ATN2 ( float_expression , float_expression )`
CEILING	Returns the smallest integer greater than, or equal to, the given numeric expression. E.g., CEILING(2.67) is 3. *Syntax:* `CEILING ( numeric_expression )`

Table 2-56 Mathematical Functions (cont.)

Math Function Name	Description and Syntax
COS	A mathematical function that returns the trigonometric cosine of the given angle (in radians) in the given expression. *Syntax:* `COS ( float_expression )`
COT	A mathematic function that returns the trigonometric cotangent of the specified angle (in radians) in the given float expression. *Syntax:* `COT ( float_expression )`
DEGREES	Returns the angle in degrees for an input angle in radians. E.g., DEGREES(PI()/2) is 90.0. *Syntax:* `DEGREES ( numeric_expression )`
EXP	Returns the exponential value of the given float expression. That is, the natural logarithm base (approx. 2.71) raised to the exponent passed as argument. E.g., EXP(1) is 2.71. *Syntax:* `EXP ( float_expression )`
FLOOR	Returns the largest integer less than or equal to the given numeric expression. E.g., FLOOR(2.67) is 2. *Syntax:* `FLOOR ( numeric_expression )`
LOG	Returns the natural logarithm of the given float expression. *Syntax:* `LOG ( float_expression )`
LOG10	Returns the base-10 logarithm of the given float expression. *Syntax:* `LOG10 ( float_expression )`
PI	Returns the constant value of PI. I.e., PI() is 3.14159. *Syntax:* `PI ( )`
POWER	Returns the value of the given expression to the specified power. *Syntax:* `POWER ( numeric_expression , y )`
RADIANS	Returns radians when a numeric expression, in degrees, is entered. *Syntax:* `RADIANS ( numeric_expression )`
RAND	Returns a random float value from 0 through 1. *Syntax:* `RAND ( [ seed ] )`
ROUND	Returns a numeric expression, rounded to the specified length or precision. *Syntax:* `ROUND ( numeric_expression , length [ , function ] )`
SIGN	Returns the positive (+1), zero (0), or negative (–1) sign of the given expression. *Syntax:* `SIGN ( numeric_expression )`

Table 2-56 Mathematical Functions (cont.)

Math Function Name	Description and Syntax
SIN	Returns the trigonometric sine of the given angle (in radians) in an approximate numeric (float) expression. *Syntax:* `SIN ( float_expression )`
SQRT	Returns the square root of the given expression. *Syntax:* `SQRT ( float_expression )`
SQUARE	Returns the square of the given expression. *Syntax:* `SQUARE ( float_expression )`
TAN	Returns the tangent of the input expression which is an angle in radians. *Syntax:* `TAN ( float_expression )`

Example:

```
SQL
SELECT  CEILING(2.13) Ceil , LOG(10) Log , LOG10(10) Log10 ,
  PI() Pi , SIN(1) Sine

Result
Ceil  Log                          Log10        Pi                      Sine
----- ------------------------------ ------------ ------------------------ ------------------------------
3     2.3025850929940459           1.0          3.1415926535897931      0.8414709848078965
```

2.6.6 Metadata Functions

A metadata function returns information about the database and database objects (Table 2-57). All are nondeterministic (see page 163).

Table 2-57 Metadata Functions

Function	Description and Syntax
@@PROCID	Returns the current stored procedure identifier (ID).
COL_LENGTH	Returns the defined length (in bytes) of a column. *Syntax:* `COL_LENGTH ( 'table' , 'column' )`

Table 2-57 Metadata Functions (cont.)

Function	Description and Syntax
COL_NAME	Returns the name of a database column given the corresponding table identification number and column identification number. *Syntax:* `COL_NAME ( table_id , column_id )`
COLUMNPROP-ERTY	Returns information about a column or procedure parameter. *Syntax:* `COLUMNPROPERTY ( id , column , property )`
DATABASE-PROP-ERTY	Returns named database property value for the given database and property name for SQL Server 7.0 and before. *Syntax:* `DATABASEPROPERTY( database , property )`
DATABASE-PROPERTYEX	Returns named database property value for the given database and property name for SQL Server 2K and later. *Syntax:* `DATABASEPROPERTYEX( database , property )`
DB_ID	Returns the database identification (ID) number. *Syntax:* `DB_ID ( [ 'database_name' ] )`
DB_NAME	Returns the database name. *Syntax:* `DB_NAME ( database_id )`
FILE_ID	Returns the file identification (ID) number for the given logical file name in the current database. *Syntax:* `FILE_ID ( 'file_name' )`
FILE_NAME	Returns the logical file name for the given file identification (ID) number. *Syntax:* `FILE_NAME ( file_id )`
FILEGROUP_ID	Returns the filegroup identification (ID) number for the given filegroup name. *Syntax:* `FILEGROUP_ID ( 'filegroup_name' )`
FILEGROUP_NA ME	Returns the filegroup name for the given filegroup identification (ID) number. *Syntax:* `FILEGROUP_NAME ( filegroup_id )`
FILEGROUP-PROPERTY	Returns the specified filegroup property value when given a filegroup and property name. *Syntax:* `FILEGROUPPROPERTY ( filegroup_name , property )`
FILEPROPERTY	Returns the specified file name property value when given a file name and property name. *Syntax:* `FILEPROPERTY ( file_name , property )`

Table 2-57 Metadata Functions (cont.)

Function	Description and Syntax
fn_listextended-property	Returns extended property values of database objects. *Syntax:* `fn_listextendedproperty ({ default | [@name =] 'property_name' | NULL } , { default | [@level0type =] 'level0_object_type' | NULL } , { default | [@level0name =] 'level0_object_name' | NULL } , { default | [@level1type =] 'level1_object_type' | NULL } , { default | [@level1name =] 'level1_object_name' | NULL } , { default | [@level2type =] 'level2_object_type' | NULL } , { default | [@level2name =] 'level2_object_name' | NULL })`
FULLTEXTCATA-LOGPROPERTY	Returns information about full-text catalog properties. *Syntax:* `FULLTEXTCATALOGPROPERTY ( catalog_name , property )`
FULLTEXTSERVI-CEPROPERTY	Returns information about full-text service-level properties. *Syntax:* `FULLTEXTSERVICEPROPERTY ( property )`
INDEX_COL	Returns the indexed column name. *Syntax:* `INDEX_COL ( 'table' , index_id , key_id )`
INDEXKEY_ PROPERTY	Returns information about the index key. *Syntax:* `INDEXKEY_PROPERTY ( tableID, indexID, keyID, property )`
INDEXPROPERTY	Returns the named index property value given a table identification number, index name and property name. *Syntax:* `INDEXPROPERTY ( table_ID , index , property )`
OBJECT_ID	Returns the database object identification number. *Syntax:* `OBJECT_ID ( 'object' )`
OBJECT_NAME	Returns the database object name. *Syntax:* `OBJECT_NAME ( object_id )`
OBJECT-PROPERTY	Returns information about objects in the current database. *Syntax:* `OBJECTPROPERTY ( id , property )`
SQL_VARIANT_ PROPERTY	Returns the base data type and other information about an sql_variant value. *Syntax:* `SQL_VARIANT_PROPERTY ( expression, property )`
TYPEPROPERTY	Returns information about a data type. *Syntax:* `TYPEPROPERTY ( type , property )`

2.6.7 Rowset Functions

A rowset is a set of rows that contain columns of data and can be used like a table in SQL. Rowsets are central objects that enable all OLE DB data providers to expose result set data in tabular form. Rowset Functions return rowsets (Table 2-58). All are nondeterministic (see page 163).

A rowset is a set of rows that contain columns of data. Rowsets are central objects that enable all OLE DB data providers to expose result set data in tabular form.

Table 2-58 Rowset Functions

Rowset Function Name	Description and Syntax
CONTAINSTABLE	Returns a table of zero or more rows after doing a full-text type query based on precise or near (fuzzy) match criteria. CONTAINSTABLE can be referenced in the FROM clause of a SELECT statement as if it were a regular table name. *Syntax:* `CONTAINSTABLE ( table , { column | * } , ' < contains_search_condition > ' [ , top_n_by_rank ] )`
FREETEXTTABLE	Returns a table of zero or more rows after doing a full-text type query based on meaning of the text. FREETEXTTABLE can be referenced in the FROM clause of a SELECT statement as if it were a regular table name. *Syntax:* `FREETEXTTABLE ( table , { column | * } , 'freetext_string' [ , top_n_by_rank ] )`
OPENDATASOURCE	Provides ad hoc connection information as part of a four-part object name without using a linked server name. See Books Online. *Syntax:* `OPENDATASOURCE ( provider_name, init_string )`
OPENQUERY	Executes the specified pass-through query on the given linked server, which is an OLE DB data source. The OPENQUERY function can be referenced in the FROM clause of a query as though it is a table name or as the target table of an INSERT, UPDATE or DELETE statement, subject to the OLE DB provider. *Syntax:* `OPENQUERY ( linked_server , 'query' )`

Table 2-58 Rowset Functions (cont.)

Rowset Function Name	Description and Syntax

OPENROWSET	This is an ad hoc method of connecting and accessing remote data using OLE DB and is an alternative to accessing tables in a linked server.
	The OPENROWSET function can be referenced in the FROM clause of a query like a table or as the target table of an INSERT, UPDATE or DELETE statement, subject to the OLE DB provider.
	Syntax:
	`OPENROWSET ( 'provider_name'`
	`, { 'datasource' ; 'user_id' ; 'password'`
	`| 'provider_string' }`
	`, { [ catalog. ] [ schema. ] object`
	`| 'query' }`
	`)`

OPENXML	Opens an XML document and returns the data as a rowset which may be used as a table in a read-only SQL statement.
	Often used with **sp_xml_preparedocument** as in the example below.
	Syntax:
	`OPENXML (`
	`idoc int [in], -- the document handle created by sp_xml_preparedocument`
	`rowpattern nvarchar[in] -- XPATH pattern to identify XML document nodes to be used as rows`
	`[, flags byte[in]] -- XML mapping:`
	`0 (default) attribute-centric,`
	`1 - attribute-centric then can use XML_ELEMENTS,`
	`2 - element-centric then can use XML_ATTRIBUTES,`
	`8 - combine XML_ELEMENTS and XML_ATTRIBUTES`
	`)[WITH (SchemaDeclaration | TableName)] -- may specify nodes to display in result set`

Example: Use OPENQUERY — requires a link be made to remote server. See page 66.

SQL

```
SELECT  TOP 2   *
FROM  OPENQUERY( CAT2_Link , 'SELECT *   FROM pubs.dbo.authors')
```

Result

au_id	au_lname	au_fname	phone	address	city	state
172-32-1176	White	Johnson	408 496-7223	10932 Bigge Rd.	Menlo Park	CA
213-46-8915	Green	Marjorie	415 986-7020	309 63rd St. #411	Oakland	CA

This example uses the link `CAT2_Link` created on page 47. That version is repeated below as it seems more direct than the OPENQUERY method above.

SQL

```
SELECT  TOP 2  *   FROM  CAT2_Link.pubs.dbo.authors
```

Result

Same result as previous.

Example: Use OPENROWSET — Does not require that a link be created first.

SQL

```
SELECT a.*
FROM    OPENROWSET('SQLOLEDB','cat\cat2';'sa';'',
```

SQL

```
'SELECT  TOP 2 * FROM pubs.dbo.authors ') AS a
```

Result

Same result as previous.

Example: Use OPENXML Create an xml document with **sp_xml_preparedocument**, then read it with OPENXML.

Create an internal representation of an XML data document and assign the document handle value to the idoc handle variable so it can be passed in to OPENXML to read.

```
DECLARE @idoc int -- Declare an int variable for the xml document handle

        EXEC sp_xml_preparedocument @idoc OUTPUT,
        '<ROOT>
        <Customer >
            <CustomerID>12</CustomerID>
            <Name>Amy Darling</Name>
            <Telephone>111-2222</Telephone>
        </Customer>
```

```
<Customer >
    <CustomerID>36</CustomerID>
    <Name> Torie Dearest </Name>
    <Telephone>333-4444</Telephone>
</Customer>
</ROOT>'
```

Do a query using OPENXML to read the desired parts of the XML data document.

SQL

```
SELECT    *
FROM OPENXML (@idoc, '/ROOT/Customer', 2)
  WITH (CustomerID  varchar(10), Name varchar(20))
```

Result

```
CustomerID      Name
----------      --------------------
2               Amy Darling
36              Torie Dearest

(2 row(s) affected)
```

With the XML data thus available at a virtual table in SQL, it may be inserted into a database table as in this example run in the same batch as the previous statements.

SQL

```
SELECT    *
INTO table1
FROM       OPENXML (@idoc, '/ROOT/Customer', 2)
  WITH (CustomerID  varchar(10), Name varchar(20))
go

SELECT *  FROM table1
```

SQL (cont.)

Result

```
CustomerID  Name
----------  --------------------
12          Amy Darling
36          Torie Dearest

(2 row(s) affected)
```

2.6.8 Security Functions

A security function returns information about users and roles. All are nondeterministic (see Table 2-59 and page 162).

Table 2-59 Security Functions

Security Function Name	Description and Syntax
fn_trace_geteventinfo	Returns information about the events traced. *Syntax:* `fn_trace_geteventinfo ( [ @traceid = ] trace_id )`
fn_trace_getfilterinfo	Returns information about the filters applied to a specified trace. *Syntax:* `fn_trace_getfilterinfo( [ @traceid = ] trace_id )`
fn_trace_getinfo	Returns information about a specified trace or existing traces. *Syntax:* `fn_trace_getinfo( [ @traceid = ] trace_id )`
fn_trace_gettable	Returns trace file information in a table format. *Syntax:* `fn_trace_gettable( [ @filename = ] filename , [ @numfiles = ] number_files )`
HAS_DBACCESS	Indicates whether the user has access to the specified database. Returns int 0 if no, 1 if yes, NULL if database name is invalid. *Syntax:* `HAS_DBACCESS ( 'database_name' )`
IS_MEMBER	Indicates whether the current user is a member of the specified Microsoft Windows NT group or Microsoft SQL Server role. Returns int 0 if no, 1 if yes, NULL if group or role is invalid. *Syntax:* `IS_MEMBER ( { 'group' | 'role' } )`
IS_SRVROLEMEMBER	Indicates whether the current or specified user login is a member of the specified server role. Returns int 0 if no, 1 if yes, NULL if role or login is invalid. *Syntax:* `IS_SRVROLEMEMBER ( 'role' [ , 'login' ] )`

Table 2-59 Security Functions (cont.)

SUSER_SID	Returns the security identification number (SID) for the user's login name. *Syntax:* `SUSER_SID ( [ 'login' ] )` **Example:** **SQL:** `SELECT SUSER_SID( 'sa')` **Result:** ---------- 0x01
SUSER_SNAME	Returns the login identification name for the current user or from the user's security identification number (SID) if specified. *Syntax:* `SUSER_SNAME ( [ server_user_sid ] )` **Example:** **SQL:** `SELECT SUSER_SNAME ( 0x1 )` **Result:** ----------- sa
USER	Allows a system-supplied value for the current user's database username to be inserted into a table when no default value is specified. *Syntax:* `USER` **Example:** **SQL:** `SELECT USER` **Result:** ------ dbo
USER_ID	Returns a user's database identification number. *Syntax:* `USER_ID ( [ 'user' ] )` **Example:** **SQL:** `SELECT USER_ID()` **Result:** ------ 1

2.6.9 String Functions — for `CHAR` or `VARCHAR` expressions

A string function for CHAR or VARCHAR expressions performs an operation on a string input value and returns a string or numeric value (Table 2-60). All are deterministic except CHARINDEX and PATINDEX (see page 162).

Table 2-60 String Functions

String Fcn	Description and Syntax
ASCII	Returns the ASCII code value of the leftmost character of a char expression. *Syntax:* `ASCII ( character_expression )` **SQL:** `PRINT ASCII('abc') -- The ASCII value of the letter "a" is 97` **Result:** 97

Table 2-60 String Functions (cont.)

String Fcn	Description and Syntax
CHAR	A string function that converts an int ASCII code to a character. Inverse of ASCII. *Syntax:* `CHAR ( integer_expression )` **SQL:** `PRINT CHAR(97) -- 97 is the ASCII value of the letter "a"` **Result:** a
CHARINDEX	Returns the starting position of expr1 in a character string expr2. Search begins with 1st character unless start_location is given and is > 1. *Syntax:* `CHARINDEX ( expr1 , expr2 [ , start_location ] )`
DIFFERENCE	Returns the difference between the SOUNDEX values of two character expressions as an integer. See Books Online. *Syntax:* `DIFFERENCE ( character_expression , character_expression )`
LEFT	Returns the leftmost *integer_expression characters* of *character_expr*. *Syntax:* `LEFT ( character_expr , integer_expression )` **SQL:** `PRINT LEFT('abcd' , 2)` **Result:** ab
LEN	Returns the number of characters (may not be the number of bytes) of the given string expression, excluding trailing blanks. *Syntax:* `LEN ( string_expression )` **SQL:** `PRINT LEN('abc')` **Result:** 3
LOWER	Returns the character_expression string in all lower case. *Syntax:* `LOWER ( character_expression )` **SQL:** `PRINT LOWER('AbCd')` **Result:** abcd
LTRIM	Returns a character expression after removing leading blanks. *Syntax:* `LTRIM ( character_expression )`
NCHAR	Returns the Unicode character with the given integer code. *Syntax:*`NCHAR ( integer_expression )`
PATINDEX	Returns the starting position of the first occurrence of a pattern in the char_expr or zero if the pattern is not found (text or character data types). *Syntax:*`PATINDEX ( '%pattern%' , char_expr )` **Example:** **SQL:** `SELECT PATINDEX( '%cd%' , 'abcde' )` **Result:** 3

Table 2-60 String Functions (cont.)

String Fcn	Description and Syntax
QUOTENAME	Returns a Unicode string with the delimiters added to make the input string a valid Microsoft SQL Server delimited identifier. *Syntax:* `QUOTENAME ( 'character_string' [ , 'quote_character' ] )`
REPLACE	Replaces all occurrences of the second given string expression in the first string expression with a third expression. *Syntax:* `REPLACE ( 'string_expr1' , 'string_expr2' , 'string_expr3' )` **Example:** **SQL:** `SELECT REPLACE ('aaaXXXbbbXXXccc' , 'XXX' , 'YYY')` **Result:** ---------- aaaYYYbbbYYYccc
REPLICATE	Repeats a character expression for a specified number of times. *Syntax:* `REPLICATE ( character_expression , integer_expression )`
REVERSE	Returns the reverse of a character expression. *Syntax:* `REVERSE ( character_expression )`
RIGHT	Returns the rightmost <integer_expr> characters of <character_expr>. *Syntax:*`RIGHT ( character_expr , integer_expr )`
RTRIM	Returns a character expression after removing trailing blanks. *Syntax:* `RTRIM ( character_expression )`
SOUNDEX	Returns a four-character (SOUNDEX) code to evaluate the similarity of two strings. *Syntax:* `SOUNDEX ( character_expression )`
SPACE	Returns a string of repeated spaces. *Syntax:* `SPACE ( integer_expression )`
STR	Returns character data converted from numeric data. *Syntax:* `STR ( float_expression [ , length [ , decimal ] ] )`
STUFF	Replaces characters in char_expr1 from start to start plus length with char_expr2. *Syntax:* `STUFF ( char_expr1 , start , length , char_expr2 )`
SUBSTRING	Returns part of a character, binary, text, or image expression. *Syntax:* `SUBSTRING ( expression , start , length )`
UNICODE	Returns the integer Unicode value for the first character of the expression. *Syntax:* `UNICODE ( 'ncharacter_expression' )`
UPPER	Returns the character_expression string in all upper case. *Syntax:* `UPPER ( character_expression )`

2.6.10 System Functions

System functions, listed in Table 2-61, perform operations and return information about values, objects and settings in Microsoft SQL Server. Some are deterministic and some are not. See list page 162.

Table 2-61 System Functions

System Function Name	Description and Syntax
@@ERROR	Returns the error number for the last Transact-SQL statement executed. *Syntax:* `@@ERROR`
@@IDENTITY	Returns the last-inserted identity value. *Syntax:* `@@IDENTITY`
@@ROWCOUNT	Returns the number of rows affected by the last statement. *Syntax:* `@@ROWCOUNT`
@@TRANCOUNT	Returns the number of active transactions for the current connection. *Syntax:* `@@TRANCOUNT`
APP_NAME	Returns the application name for the current session if set. *Syntax:* `APP_NAME ( )`
CASE expression	Evaluates a list of conditions and returns one of multiple possible result expressions. See explanation and examples, page 165.

CAST and CONVERT	Explicitly converts an expression of one data type to another. CAST and CONVERT provide similar functionality. See explanation and examples, page 167.
	Syntax:
	`CAST ( expression AS data_type )`
	`CONVERT ( data_type [ ( length ) ] , expression [ , style ] )`

COALESCE	Returns the first nonnull expression among its arguments. *Syntax:* `COALESCE ( expression [ ,...n ] )`
COLLATIONPROPERTY	Returns the property of a given collation. *Syntax:* `COLLATIONPROPERTY( collation_name, property)`
CURRENT_TIMESTAMP	Returns the current date and time. Equivalent to GETDATE(). *Syntax:* `CURRENT_TIMESTAMP`
CURRENT_USER	Returns the current user. This function is equivalent to USER_NAME(). *Syntax:* `CURRENT_USER`

Table 2-61 System Functions (cont.)

System Function Name	Description and Syntax
DATALENGTH	Returns the number of bytes used to represent an expression. *Syntax:* `DATALENGTH ( expression )`
fn_helpcollations	Returns a list of all the collations supported by SQL Server 2K. *Syntax:* `fn_helpcollations ()`
fn_servershareddrives	Returns the names of shared drives used by the clustered server. *Syntax:* `fn_servershareddrives()`

fn_virtualfilestats	Returns I/O statistics for database files, including log files.
	Syntax:
	`fn_virtualfilestats ( [@DatabaseID=] database_id ,`
	`[ @FileID = ] file_id )`

FORMATMESSAGE	Constructs a message from an existing message in sysmessages. The functionality of FORMATMESSAGE resembles that of the RAISERROR statement; however, RAISERROR prints the message immediately. FORMATMESSAGE returns the edited message for further processing. *Syntax:* `FORMATMESSAGE ( msg_number , param_value [ ,...n ] )`
GETANSINULL	Returns the effective default nullability for the database for this session. *Syntax:* `GETANSINULL ( [ 'database' ] )`
HOST_ID	Books Online says this returns the client workstation identification number. This value appears to be the process id for each client program from the client host. Thus the value differs for each client program. *Syntax:* `HOST_ID ( )`
HOST_NAME	Returns the client workstation name. This name is the same for each connection from the client host. *Syntax:* `HOST_NAME ( )`
IDENT_CURRENT	Returns the last identity value generated for a specified table in any session and any scope. *Syntax:* `IDENT_CURRENT('table_name')`
IDENT_INCR	Returns the numeric increment value specified during the creation of an identity column in a table or view that has an identity column. *Syntax:* `IDENT_INCR ( 'table_or_view' )`
IDENT_SEED	Returns the numeric seed value specified during the creation of an identity column in a table or a view that has an identity column. *Syntax:* `IDENT_SEED ( 'table_or_view' )`

Table 2-61 System Functions (cont.)

System Function Name	Description and Syntax
IDENTITY (Function)	Is used only in a SELECT statement with an INTO table clause to insert an identity column into a new table. The IDENTITY function is similar to the IDENTITY property used with CREATE TABLE. *Syntax:* `IDENTITY ( data_type [ , seed , increment ] ) AS column_name` **Example:** **SQL:** `SELECT IDENTITY(INT) As id , name INTO newemp FROM emp`
ISDATE	Returns 1 if the expression is a valid date, 0 if not. *Syntax:* `ISDATE ( expression )`
ISNULL	If *expr1* is NULL, it is replaced with *expr2*. That is, IF *expr1* IS NOT NULL returns *expr1* ELSE returns *expr2* *Syntax:* `ISNULL ( expr1 , expr2 )` **Example:** **SQL:** `SELECT ISNULL( 'Hello', 'Null') , ISNULL (NULL, 'Null word')` **Result:** ---------- ---------- Hello Null word
ISNUMERIC	Returns 1 if the expression is a valid numeric type, 0 if not. *Syntax:* `ISNUMERIC ( expression )`
NEWID	Creates a unique value of type uniqueidentifier. *Syntax:* `NEWID ( )`
NULLIF	Returns NULL if the two expressions are equivalent. If not equivalent, returns the first expression. See Books Online for meaningful example. *Syntax:* `NULLIF ( expression , expression )`
PARSENAME	Returns the specified part of an object name. Parts of an object that can be retrieved are the object name, owner name, database name, and server name. Note: This function does not indicate whether or not the specified *object* exists. It just returns the specified piece of the given *object name*. *Syntax:* `PARSENAME ( 'object_name' , object_piece )` object_piece (integer) Meaning 1 Object name 2 Owner name 3 Database name 4 Server name

Table 2-61 System Functions (cont.)

PERMISSIONS	Returns a value containing a bitmap that indicates the statement, object or column permissions for the current user. *Syntax:* `PERMISSIONS ( [ objectid [ , 'column' ] ] )`
ROWCOUNT_BIG	Returns the number of rows affected by the last statement executed. This function operates like @@ROWCOUNT, except that the return type of ROWCOUNT_BIG is bigint. *Syntax:* `ROWCOUNT_BIG ( )`
SCOPE_IDENTITY	Returns the last IDENTITY value inserted into an IDENTITY column in the same scope. A scope is a module — a stored procedure, trigger, function or batch. Thus, two statements are in the same scope if they are in the same stored procedure, function or batch. *Syntax:* `SCOPE_IDENTITY( )`
SERVERPROPERTY	Returns property information about the server instance. *Syntax:* `SERVERPROPERTY ( propertyname )`
SESSIONPROPERTY	Returns the SET options settings of a session. *Syntax:* `SESSIONPROPERTY ( option )`
SESSION_USER	Returns the username of the current user. May be used as a column DEFAULT in CREATE TABLE to insert the name of the user executing an INSERT statement. See example below under SYSTEM_USER. *Syntax:* `SESSION_USER`
STATS_DATE	Returns the date and time that the statistics for the specified index were last updated. See example page 297. *Syntax:* `STATS_DATE ( table_id , index_id )`
SYSTEM_USER	Returns the system username of the current user. May be used as a column DEFAULT in CREATE TABLE to insert the name of the user executing an INSERT statement. *Syntax:* `SYSTEM_USER` **Example:** **SQL:** `SELECT SESSION_USER As Sess , SYSTEM_USER As Sys` **Result:** Sess Sys ----- ------- dbo sa
USER_NAME	Returns a user database username from a given identification number. *Syntax:* `USER_NAME ( [ id ] )`

2.6.11 System Statistical Functions

A system statistical function, returns statistical information about the system. See Table 2-62 for details. All are nondeterministic (see page 162).

Table 2-62 System Statistical Functions

Function Name	Description and Syntax
@@CONNECTIONS	Returns the number of connections, or attempted connections, since Microsoft SQL Server was last started.
@@CPU_BUSY	Returns the time in milliseconds (based on the resolution of the system timer) that the CPU has spent working since Microsoft SQL Server was last started.
@@IDLE	Returns the time in milliseconds (based on the resolution of the system timer) that Microsoft SQL Server has been idle since last started.
@@IO_BUSY	Returns the time in milliseconds (based on the resolution of the system timer) that Microsoft SQL Server has spent performing input and output operations since it was last started.
@@PACK_RECEIVED	Returns the number of input packets read from the network by Microsoft SQL Server since last started.
@@PACK_SENT	Returns the number of output packets written to the network by Microsoft-SQL Server since last started.
@@PACKET_ERRORS	Returns the number of network packet errors that have occurred on Microsoft SQL Server connections since SQL Server was last started.
@@TIMETICKS	Returns the number of microseconds per tick.
@@TOTAL_ERRORS	Returns the number of disk read/write errors encountered by Microsoft SQL Server since last started.
@@TOTAL_READ	Returns the number of disk reads (not cache reads) by Microsoft SQL Server since last started.
@@TOTAL_WRITE	Returns the number of disk writes by Microsoft SQL Server since last started.
fn_virtualfilestats	Returns I/O statistics for database files, including log files. *Syntax:* `fn_virtualfilestats ( [@DatabaseID=]database_id , [ @FileID = ] file_id )`

2.6.12 Text and Image Functions and Statements

Text and image functions, listed in Table 2-63, perform an operation on a text or image input value or column and return information about the value. All are nondeterministic (see page 162).

Table 2-63 Text and Image Functions

Function Name	Description and Syntax
DATALENGTH	Returns the number of bytes used to represent any expression. *Syntax:* `DATALENGTH ( expression )`
PATINDEX	Returns the starting position of the first occurrence of a pattern in a specified expression or zero if the pattern is not found. All text and character data types. *Syntax:* `PATINDEX ( '%pattern%' , expression )`
SUBSTRING	Returns part of a character, binary, text, or image expression. See Books Online. *Syntax:* `SUBSTRING ( expression , start , length )`
TEXTPTR	Returns the text-pointer value that corresponds to a text, ntext, or image column in varbinary format. The retrieved text pointer value can be used in READTEXT, WRITETEXT, and UPDATETEXT statements. *Syntax:* `TEXTPTR ( column )`
TEXTVALID	Returns 1 if a given text, ntext, or image pointer is valid, 0 if not. *Syntax:* `TEXTVALID ( 'table.column' , text_ ptr )`

See Text examples on page 109.

Text and image statements are summarized in Table 2-64.

Table 2-64 Text and Image Statements

Statement Name	Description and Syntax
READTEXT	Reads text, ntext, or image values from a text, ntext, or image column, starting from a specified offset and reading the specified number of bytes. *Syntax:* `READTEXT { table.column text_ptr offset size } [ HOLDLOCK ]`
SET TEXTSIZE	Specifies the size of text and ntext data returned with a SELECT statement. *Syntax:* `SET TEXTSIZE { number }`
UPDATETEXT	Updates an existing text, ntext, or image field. Use UPDATETEXT to change only a portion of a text, ntext or image column in place.

Table 2-64 Text and Image Statements (cont.)

Statement Name	Description and Syntax
WRITETEXT	Permits nonlogged, interactive updating of an existing text, ntext or image column. This statement completely overwrites any existing data in the column it affects. WRITETEXT cannot be used on text, ntext and image columns in views. *Syntax:* `WRITETEXT { table.column text_ptr } [ WITH LOG ] { data }`

2.6.13 Deterministic and Nondeterministic Functions

All functions are either deterministic or nondeterministic. Deterministic functions always return the same result any time they are called with the same input values. For example, ABS(-2) always returns 2. Nondeterministic functions may return different results each time they are called even though input values are the same. For example, GETDATE() returns a different result each time it's called.

Indexed views or indexes on computed columns cannot include nondeterministic functions. An index cannot be created on a view which references any nondeterministic functions. An index cannot be created on a computed column if the *computed_column_expression* references any nondeterministic functions.

2.6.13.1 Listing of Deterministic and Nondeterministic Functions

Aggregate built-in functions (page 139) are all deterministic. String built-in functions (page 154) are all deterministic except CHARINDEX and PATINDEX. Tables 2-65 thorugh 2-68 identify characteristics of many functions.

Always Deterministic The functions in Table 2-65 are always deterministic.

Table 2-65 Deterministic Functions

ABS	COS	EXP	NULLIF	SIN
ACOS	COT	FLOOR	PARSENAME	SQUARE
ASIN	DATALENGTH	ISNULL	PI	SQRT
ATAN	DATEADD	ISNUMERIC	POWER	TAN
ATN2	DATEDIFF	LOG	RADIANS	YEAR
CEILING	DAY	LOG10	ROUND	
COALESCE	DEGREES	MONTH	SIGN	

The System Functions listed in Table 2-66 are deterministic.

Table 2-66 Deterministic System Functions

CASE expression	COALESCE	DATALENGTH	fn_helpcollations	ISNULL
ISNUMERIC	NULLIF	PARSENAME		

Sometimes Deterministic These functions, listed in Table 2-67, are not always deterministic but can be used in indexed views or indexes on computed columns when they are specified in a deterministic manner.

Table 2-67 Sometimes Deterministic Functions

Function	Comments
CAST	Deterministic unless used with **datetime**, **smalldatetime** or **sql_variant**.
CONVERT	Deterministic unless used with **datetime**, **smalldatetime** or **sql_variant**. The **datetime** and **smalldatetime** data types are deterministic if style parameter is given.
CHECKSUM	Deterministic, with the exception of CHECKSUM(*).
DATEPART	Deterministic except when used as DATEPART (dw, *date*). The value returned by **dw**, weekday, depends on the value set by SET DATEFIRST.
ISDATE	Deterministic only if used with the CONVERT function, the CONVERT style param-e ter is specified and style is not equal to 0, 100, 9 or 109.
RAND	RAND is deterministic only when a *seed* parameter is specified.

Never Deterministic The System and Built-in Functions in Table 2-68 are always nondeterministic.

Table 2-68 Nondeterministic Functions

@@ERROR	fn_servershareddrives	IDENT_INCR	SESSIONPROPERTY
@@IDENTITY	fn_virtualfilestats	IDENT_SEED	STATS_DATE
@@ROWCOUNT	FORMATMESSAGE	IDENTITY	SYSTEM_USER
@@TRANCOUNT	GETANSINULL	NEWID	TEXTPTR
APP_NAME	GETDATE	PERMISSIONS	TEXTVALID
COLLATIONPROPERTY	GETUTCDATE	ROWCOUNT_BIG	USER_NAME
CURRENT_TIMESTAMP	HOST_ID	SCOPE_IDENTITY	

Table 2-68 Nondeterministic Functions (cont.)

CURRENT_USER	HOST_NAME	SERVERPROPERTY	
DATENAME	IDENT_CURRENT	SESSION_USER	

As discussed earlier, all configuration, cursor, meta data, rowset, security, and system statistical functions are nondeterministic. Functions that call extended stored procedures are nondeterministic because the extended stored procedures can cause side effects on the database.

2.6.14 CASE Expression

CASE can be considered an expression or a function because it evaluates to a single scalar value of the same data type as the input expression. CASE has two formats: simple CASE and searched CASE.

Simple CASE compares the input expression to a series of simple expressions.

```
CASE input-expression WHEN match-expression THEN result
                      [ WHEN match-expression THEN result ]
                      ...
                      [ELSE result]
END
```

Searched CASE evaluates a series of Boolean expressions to determine the result.

```
CASE                 WHEN Boolean-condition THEN result
                     [ WHEN Boolean-condition THEN result ]
                     ...
                     [ELSE result2]
END
```

2.6.14.1 Example of Simple CASE

Consider Table 2-69, which has a column containing a car manufacturer abbreviation.

Table 2-69 Autos

Make	Manufacturer	...	...
Buick	GM		
Quattro	Au		

Table 2-69 Autos (cont.)

Jeep	DC		
Sebring	DC		

The following query uses CASE to convert the manufacturer abbreviation to the full name.

SQL

```
SELECT Make, CASE Manufacturer
  WHEN 'GM' THEN 'General Motors'
  WHEN 'Au'THEN 'Audi'
  WHEN 'DC'THEN 'Daimler-Chrysler'
  ELSE 'Manufacturer not found'
END   As Manufacturer
FROM Autos;
```

Result

```
Make        Manufacturer
----------  --------------------
Buick       General Motors
Quattro     Audi
Jeep        Daimler-Chrysler
Sebring     Daimler-Chrysler
```

2.6.14.2 Example of Searched CASE

This form of CASE can be used for inequalities, as in the example, as well as equalities. Consider Table 2-70, for which we want to do a query that assigns letter grades.

Table 2-70 Grades

Student	Grade	Major	...
Torie	87	Counselling	
James	76	Dog Husbandry	
Amy	93	Tae Kwon Do	

Table 2-70 Grades (cont.)

Student	Grade	Major	...
Tim	82	Jet Skiing	
Ina	98	Flower Gardening	

SQL

```
SELECT Student, CASE   WHEN Grade > 90 THEN 'A'
                       WHEN Grade > 80 THEN 'B'
                       WHEN Grade > 70 THEN 'C'
                       WHEN Grade > 60 THEN 'D'
                       ELSE 'F'
END    As  LetterGrade
FROM Grades
ORDER BY Student;
```

Result

```
Student     LetterGrade
----------  --------------
Amy         A
Ina         A
James       C
Tim         B
Torie       B
```

2.6.15 CAST and CONVERT

Both CAST and CONVERT functions are used to explicitly convert a value from one data type to another data type. CAST and CONVERT provide similar functionality but only CAST complies with ANSI SQL-92 and -99. CAST and CONVERT may be used anywhere a scalar valued expression may occur in an SQL statement.

CAST Syntax

```
CAST ( expression AS datatype )
```

CONVERT Syntax

```
CONVERT (data_type[(length)], expression [, style])
```

Example: Error

```
SQL
SELECT Make, CASE Manufacturer
    WHEN 'GM' THEN 'General Motors'
    WHEN 'Au'THEN 'Audi'
    WHEN 'DC'THEN 'Daimler-Chrysler'
    ELSE 'Manufacturer not found'
   END   As Manufacturer
FROM Autos;

Result
Server: Msg 241, Level 16, State 1, Line 2
Syntax error converting datetime from character string.

SQL SELECT  'Today is ' + GETDATE()   -- Error, incompatible data types
```

CAST Example: CAST is ANSI standard which makes it more portable than CONVERT.

```
SQL
SELECT  'Today is ' + CAST( GETDATE() AS CHAR ) -- Okay

Result
----------------------------------------------
Today is Sep  2 2001  2:14PM.
```

CONVERT Example:

```
SQL
SELECT  'Today is ' + CONVERT( CHAR , GETDATE() )  -- Okay

Result
----------------------------------------------
Today is Sep  2 2001  2:14PM.
```

2.7 SYSTEM STORED PROCEDURES AND DBCC

System stored procedures and extended procedures are built-in commands that perform a variety of tasks to do database administration and to provide information.

2.7.1 System Stored Procedures

There are hundreds of system stored procedures and extended procedures. Some individual stored procedures will be described as needed throughout the book. You may check the index to see if a given stored procedure is described in the book. A complete list of system stored procedures may be found in Appendix A. Metadata stored procedures are discussed on p. 376. Additional details may be found by looking each up by name in Books Online.

In general, system stored procedure names start with sp_ and return results within SQL Server. Extended stored procedure names start with xp_ and require doing work outside of SQL Server such as making calls to the operating system or an external process.

Some useful system stored procedures and extended procedures are summarized in Table 2-71.

Table 2-71 System Stored Procedures

sp_configure [*settingname* [, *settingvalue*]]	No arguments—lists all current server settings in five columns name minimum maximum config_value run_value where config_value is value configured but may not yet be effective until server restarts or "reconfigure" command is executed. 1 argument—shows its value; 2 arguments—sets its value See page 177 for discussion and examples.
sp_dboption	Included for backward compatibility. Use ALTER DATABASE.
sp_help [*objname*]	No arguments—lists all objects in the current database. 1 argument—reports information about the database object in the current database.
sp_helpdb *dbname*	Reports information about a specified database or all databases.
sp_helpindex *tablename*	Reports indexes on the table or view.
sp_helptext *objname*	Prints the code text of a rule, a default or an unencrypted stored procedure, user-defined function, trigger or view.
sp_lock [*spid*]	Reports locks held by all processes or by spid (page 140).

Table 2-71 System Stored Procedures (cont.)

sp_who	Reports all current users and processes connected to SQL Server.
xp_cmdshell	Executes a given command string as an operating system command shell and returns any output as rows of text.
xp_grantlogin	Grants a Microsoft Windows NT group or user access to Microsoft SQL Server.
xp_revokelogin	Revokes access from a Microsoft Windows NT group or user to Microsoft SQL Server.

Stored procedures are executed using the following syntax.

Syntax

```
[EXEC[UTE]]    sp_stored_procedure_name
```

EXEC or EXECUTE keyword is optional if this is the first statement in a batch.

2.7.1.1 The sp_ Procedures

The stored procedures provided by SQL server that begin with sp_ are predefined to provide administrative functions for managing SQL Server and displaying information about databases and users. Examples include the following: **sp_helpdb**, **sp_help**, **sp_configure**. See a complete list in Appendix A.

Example: Use **sp_helpdb** to display size and file information about pubs database.

SQL

```
EXEC  sp_helpdb  pubs
```

Result

```
name     db_size     owner    dbid   created                 ...
-------  ----------  -------- ------ ----------              ...
pubs     2.00 MB     sa       5      Aug  6 2000             ...

name     fileid   filename                                              ...
-------  ------   ----------                                            ...
pubs     1        C:\Program Files\Microsoft SQL Server\MSSQL\data\pubs.mdf      ...
pubs_log 2        C:\Program Files\Microsoft SQL Server\MSSQL\data\pubs_log.ldf  ...
```

2.7.1.2 The xp_ Procedures—Provided

The procedures that begin with xp_, called extended stored procedures, allow creation of external routines in a programming language such as C.

They are used to do work outside of SQL Server, such as accessing the registry, etc. A competent user may create his or her own extended stored procedures with **sp_addextendedproc**. Some examples include the following: **xp_cmdshell**, **xp_grantlogin**, **xp_revokelogin**. For more details, see Books Online Index: xp_ .

Example: Use **xp_cmdshell** to show contents of C:\ directory on server machine.

SQL

```
EXEC master..xp_cmdshell   'dir/w  C:\'
```

Result

```
output
-------------------------------------------------------------------
Volume in drive C is AMY_C
Volume Serial Number is 2CA0-6A55
Directory of C:\
boot.ini    [Documents and Settings]    [HySnap]    [Inetpub]    [WINNT]
1 File(s)          189 bytes
4 Dir(s)   9,494,585,344 bytes free
```

Example: Display the names of database objects in the **pubs** database.

SQL

```
USE pubs
go

EXEC sp_help      -- List all database objects in pubs database
```

SQL (cont.)

Result

```
Name                                Owner                Object_type
----------------------------------- -------------------- ---------------
titlevie                            dbo                  view
authors                             dbo                  user table
discounts                           dbo                  user table
employee                            dbo                  user table
...
titleauthor                         dbo                  user table
titles                              dbo                  user table
syscolumns                          dbo                  system table
syscomments                         dbo                  system table
...
PK__jobs__117F9D94                  dbo                  primary key cns
PK_emp_id                           dbo                  primary key cns
UPK_storeid                         dbo                  primary key cns
UPKCL_auidind                       dbo                  primary key cns
...
FK__discounts__stor___0F975522      dbo                  foreign key cns
...
DF__authors__phone__78B3EFCA        dbo                  default (maybe cns)
...
CK__authors__au_id__77BFCB91        dbo                  check cns
...

User_type   Storage_type  Length  Prec  Scale  Nullable  Default_name  Collation
----------- ------------- ------- ----- ------ --------- ------------- ------------------------------------
empid       char          9       9     NULL   no        none          SQL_Latin1_General_CP1_CI_AS
id          varchar       11      11    NULL   no        none          SQL_Latin1_General_CP1_CI_AS
tid         varchar       6       6     NULL   no        none          SQL_Latin1_General_CP1_CI_AS
```

Example: Display the metadata for the authors table of the **pubs** database including column names.

SQL

```
EXEC sp_help   authors      -- Show structure and
  properties of "authors" table in pubs database
```

SQL (cont.)

Result

```
Name        Owner   Type        Created_datetime
----------  ------- ----------  -----------------------
authors     dbo     user table  2000-08-06 01:33:52.123

Column_name Type    Computed    Length    Prec   Scale  Nullable
----------- ------- ----------- --------- ------ ------- -----------
au_id       id      no          11                       no
au_lname    varchar no          40                       no
au_fname    varchar no          20                       no
...
```

2.7.2 DBCC

The initials DBCC stand for Database Console Command (a k a Database Consistency Checker before SQL Server 2K). DBCC statements check the physical and logical consistency of a database. Many DBCC statements can fix detected problems.

The four categories of DBCC statements and descriptions are shown in Tables 2-72 through 2-75 below. See Books Online for details.

2.7.2.1 DBCC Maintenance Statements

The commands listed in Table 2-72 will help you perform maintenance tasks on a database, index or filegroup.

Table 2-72 DBCC Maintenance Statements

DBCC DBREINDEX	Rebuilds one or more indexes for a table in the specified database.
DBCC DBREPAIR	Drops a damaged database. NOTE: DBCC DBREPAIR is included in Microsoft SQL Server 2000 for backward compatibility only and may not appear in future versions. DROP DATABASE is recommended to drop damaged databases.
DBCC INDEXDEFRAG	Defragments indexes of the specified table or view.
DBCC SHRINKDATABASE	Shrinks the size of the data files in the specified database.
DBCC SHRINKFILE	Shrinks the size of the specified data file or log file for the related database.
DBCC UPDATEUSAGE	Reports and corrects inaccuracies in the **sysindexes** table. This may result in incorrect space usage reports by **sp_spaceused**.

2.7.2.2 DBCC Miscellaneous Statements

The commands in Table 2-73 will help you do miscellaneous tasks such as enabling row-level locking or removing a DLL from memory.

Table 2-73 DBCC Miscellaneous Statements

DBCC dllname (FREE)	Unloads the specified extended stored procedure dynamic-link library (DLL) from memory.
DBCC HELP	Returns syntax information for the specified DBCC statement.
DBCC PINTABLE	Marks a table to be pinned, which means Microsoft SQL Server does not flush the pages for the table from memory.
DBCC ROWLOCK	Does not affect the locking behavior of SQL Server 2K. It is included for backward compatibility for SS 6.5 scripts and may not appear in future versions.
DBCC TRACEOFF	Disables the specified trace flag(s).
DBCC TRACEON	Turns on (enables) the specified trace flag.
DBCC UNPINTABLE	Marks a table as unpinned. After a table is marked as unpinned, the table pages in the buffer cache can be flushed.

2.7.2.3 DBCC Status Statements

The commands listed in Table 2-74 allow you to perform status checks.

Table 2-74 DBCC Status Statements

DBCC INPUTBUFFER	Displays the last statement sent from a client to Microsoft SQL Server.
DBCC OPENTRAN	Displays information about the oldest active transaction and the oldest distributed and nondistributed replicated transactions, if any, within the specified database. Results are displayed only if there is an active transaction or if the database contains replication information. An informational message is displayed if there are no active transactions.
DBCC OUTPUTBUFFER	Returns the current output buffer in hexadecimal and ASCII format for the specified system process ID (SPID).
DBCC PROCCACHE	Displays information in a table format about the procedure cache.
DBCC SHOWCONTIG	Displays fragmentation information for the data and indexes of the specified table.
DBCC SHOW_STATISTICS	Displays the current distribution statistics for the specified target on the specified table.

Table 2-74 DBCC Status Statements (cont.)

DBCC SQLPERF	Provides statistics about the use of transaction-log space in all databases.
DBCC TRACESTATUS	Displays the status of trace flags.
DBCC USEROPTIONS	Returns the SET options active (set) for the current connection.

2.7.2.4 DBCC Validation Statements

The commands shown in Table 2-75 allow you to perform validation operations on a database, table, index, catalog, filegroup, system tables or allocation of database pages.

Table 2-75 DBCC Validation Statements

DBCC CHECKALLOC	Checks consistency of disk space allocation structures of specified database.
DBCC CHECKCATALOG	Checks for consistency in and between system tables in specified database.
DBCC CHECKCONSTRAINTS	Checks integrity of a specified constraint or all constraints on the specified table.
DBCC CHECKDB	Checks the allocation and structural integrity of all the objects in the specified database.
DBCC CHECKFILEGROUP	Checks the allocation and structural integrity of all tables in the current database in the specified filegroup.
DBCC CHECKIDENT	Checks the current identity value for the specified table and, if needed, corrects the identity value.
DBCC CHECKTABLE	Checks the integrity of the data, index, **text**, **ntext** and **image** pages for the specified table or indexed view.
DBCC NEWALLOC	DBCC NEWALLOC is identical to DBCC CHECKALLOC which is recommended. DBCC NEWALLOC is included in SS 2000 for backward compatibility and may not appear in future versions.

2.8 SERVER, DATABASE AND SESSION SETTINGS

Server configuration settings, database configuration settings and session (or connection) property settings in some cases interact and in some cases are dis-

jointed. Because some settings interact, I have found it less confusing to consider them all together.

2.8.1 Settings Overview

Most people can live a long and happy life without delving into the morass of these settings. Microsoft has done an excellent job of designing the database engine to set appropriate default values and of self-tuning to keep performance at a peak for most applications. Nonetheless, I think they need to clean up the interfaces for setting and reading the settings (see Author's Opinion below).

Generally speaking, server settings are of interest mainly to database administrators and most should be used only by experienced users and then on a test machine first. I'm not likely to change the number of "max worker threads" or "nested triggers," but if you have a reason and know what you're doing, we'll show you how. The rest of us will defer. ;-)

For database settings, one is likely to occasionally need to a change a database from MULTI_USER to RESTRICTED_USER in order to do maintenance. One may also want to set a specific database to READ_ONLY if its use does not require making data changes.Most database settings are best left as the default unless there is a specific reason to do otherwise.

Most session (connection) settings are also best left as the default values unless one has a specific reason to make a change. Exceptions include the occasional guidance given for using OLE DB or ODBC library call to set a specific option to a certain value. In these cases, I just follow the guidance without asking questions.

Session settings that can be quite useful for debugging or performance testing include NOEXEC, NOCOUNT, PARSEONLY, SHOWPLAN_*xxx,* STATISTICS *xxx*, etc. These options are well worth studying and testing to see the information they provide.

Having an understanding of the differences between server, database and session configuration settings will facilitate your programming. The major differences are listed below.

2.8.1.1 Server Configuration

Server settings affect server-wide settings and some database settings. Methods to see and assign server settings are (see details page 178):

- **sp_configure** system stored procedure (sets all options)
- **Enterprise Manager** (sets only the most commonly used options)

2.8.1.2 Database Configuration

Database settings affect database and default settings for connections to the database. Methods to see and assign database settings are (see details page 187):

- **ALTER DATABASE** with a **SET** clause — Set all db settings. See page 188.
- **DATABASEPROPERTYEX(** *'dbname'* **,** *'propertykeyword'* **)** — Read db settings. See page 195.
- **Enterprise Manager** — Set primary settings only
- **EXEC sp_dboption** — obsolete (may not be in new versions). Use ALTER DATABASE.

2.8.1.3 Session (Connection) Configuration

New sessions inherit server and database settings and the user may change some. Methods to see and assign session settings are (see details page 202):

- **SET** — Set all session options. See page 204.
- **SELECT @@OPTIONS** — Read all session options. See page 208.
- **DBCC USEROPTIONS** — Read all session options. See page 210.
- **SELECT SESSIONPROPERTY ('*option*')** — Read all session options. See page 211.

Session settings for Query Analyzer can also be read and set from its "Query" menu: "Query — Current Connection Properties."

Author's Opinion The subject of Server, Database and Session settings on SQL Server is overly confusing and needs cleanup work by Microsoft to make it easier to understand and manage. For example, why must one use ALTER DATABASE pubs SET READ_ONLY, or READ_WRITE to change whether a database is updateable, but have to use SELECT DATABASEPROPERTYEX ('pubs' , 'Updateability') to read the current setting?

And notice that **pubs** in the first statement must have no quotation marks and in the second statement it must have them. The now-out-of-favor **sp_dboption** at least had a very consistent interface for changing and reading settings.

Utilities to read current session settings also need cleanup work. @@OPTION is relatively complete but a bit awkward to use. SESSIONPROPERTY uses consistent keywords with SET, but only covers seven of them. DBCC USEROPTIONS only shows the ON settings, which is fine, but it doesn't report on all of the SET options. Oh, well!

2.8.1.4 Server Configuration Settings

Figure 2-3 shows a brief summary of the settings for server configuration. The details of these settings are in the sections that follow.

2.8.1.5 sp_configure

Use **sp_configure** to display or change global configuration settings for the current server. Table 2-76 summarizes the accessible settings.

Syntax

```
sp_configure [ [ @configname = ] 'name'      [ , [
@configvalue = ] 'value' ] ]
```

sp_configure may be executed with 0, 1 or 2 arguments:

- 0 arguments: Lists all configuration setting names addressable with **sp_configure**
- 1 argument: Displays the current setting for the configuration name specified
- 2 arguments: Sets the specified configuration name to the specified value

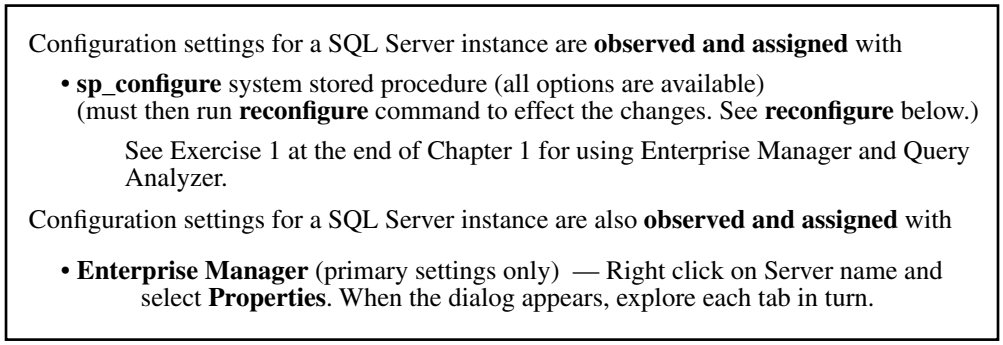

Figure 2-3 Summary of Server Configuration Statements

Table 2-76 Server Configuration Settings Accessible with **sp_configure**

sp_configure Configuration Option	Minimum	Maximum	Default	Requires 'show advanced options'	Requires Server Stop and Restart
affinity mask	0	2147483647	0	Yes	Yes
allow updates	0	1	0		
awe enabled	0	1	0	Yes	Yes

Table 2-76 Server Configuration Settings Accessible with **sp_configure** (cont.)

sp_configure Configuration option	Minimum	Maximum	Default	Requires 'show advanced options'	Requires Server Stop and Restart
c2 audit mode	0	1	0	Yes	Yes start audit, No stop audit
cost threshold for parallelism	0	32767	5	Yes	
cursor threshold	1	2147483647	–1	Yes	
default full-text language	0	2147483647	1033	Yes	
default language	0	9999	0		
fill factor (%)	0	100	0	Yes	Yes
index create memory (KB)	704	2147483647	0	Yes	
lightweightpooling	0	1	0	Yes	Yes
locks	5000	2147483647	0	Yes	Yes
max degree of parallelism	0	32	0	Yes	
max server memory (MB)	4	2147483647	2147483647	Yes	
max text repl size (B)	0	2147483647	65536		
max worker threads	32	32767	255	Yes	
media retention	0	365	0	Yes	Yes
min memory per query (KB)	512	2147483647	1024	Yes	
min server memory (MB)	0	2147483647	0	Yes	
nested triggers	0	1	1		
network packet size (B)	512	65536	4096	Yes	

Table 2-76 Server Configuration Settings Accessible with **sp_configure** (cont.)

sp_configure Configuration option	Minimum	Maximum	Default	Requires 'show advanced options'	Requires Server Stop and Restart
open objects	0	2147483647	0	Yes	Yes
priority boost	0	1	0	Yes	Yes
query governor cost limit	0	2147483647	0	Yes	
query wait (s)	–1	2147483647	–1	Yes	
recovery interval (min)	0	32767	0	Yes	
remote access	0	1	1		Yes
remote login timeout (s)	0	2147483647	20		
remote proc trans	0	1	0		
remote query timeout (s)	0	2147483647	600		
scan for startup procs	0	1	0	Yes	Yes
set working set size	0	1	0	Yes	Yes
show advanced options	0	1	0		
two digit year cutoff	1753	9999	2049	Yes	
user connections	0	32767	0	Yes	Yes
user options (See page 202)	0	32767	0		

To see the listing and current settings, execute **sp_configure** with no arguments. By default only a partial listing is given unless show advanced options is enabled.

```
SQL
EXEC sp_configure  -- Lists common configuration options.
  Enable 'advanced options' to see all.
```

SQL (cont.)

Result

name	minimum	maximum	config_value	run_value
allow updates	0	1	0	0
default language	0	9999	0	0
max text repl size (B)	0	2147483647	65536	65536
nested triggers	0	1	1	1
remote access	0	1	1	1
remote login timeout (s)	0	2147483647	20	20
remote proc trans	0	1	0	
remote query timeout (s)	0	2147483647	600	600
show advanced options	0	1	0	0
user options	0	32767	0	0

For a description of each item see Books Online: Setting Configuration Options.

2.8.1.6 `sp_configure` SHOW ADVANCED OPTIONS

To see all **sp_configure** options, not just the basic ones, enable show advanced options as shown here.

Example:

SQL

```
EXEC sp_configure 'show advanced options' , 1
RECONFIGURE  -- Must run this to make the change effective .
```

2.8.1.7 `sp_configure` USER OPTIONS

The **sp_configure USER OPTIONS** value is a single integer which is a bit-set specifying global defaults for 15 settings that affect each user's session (connection). A user may override each setting using the SET statement.

See discussion and examples of **sp_configure user options** on page 202.

Example:

SQL

```
EXEC sp_configure  -- Now lists ALL 36 configuration options.
```

SQL (cont.)

Result

name	minimum	maximum	config_value	run_value
affinity mask	0	2147483647	1	1
...				
user options	0	32767	0	0

2.8.1.8 When Do `sp_configure` Changes Become Effective?

Here is the short answer to this question: They become effective when **run_value** matches **config_value**, which depends on the option.

- All **sp_configure** changes need **RECONFIGURE** to be run to become effective.
  - Two options ('allow updates' and 'recovery interval') sometimes require RECONFIGURE WITH OVERRIDE to be run (see RECONFIGURE below).
- Some options also **require server stop and restart** as indicated in Table 2-76. The following do not need server stop and restart.

allow updates
cost threshold for parallelism
cursor threshold
index create memory (KB)
max degree of parallelism
max server memory (MB)
max text repl size (B)
max worker threads
min memory per query (KB)
min server memory (MB)
network packet size (B)
query governor cost limit
query wait (s)
recovery interval (min)
remote login timeout (s)
remote proc trans
remote query timeout (s)
show advanced options
user options
default full-text language?
default language?
nested triggers?
two digit year cutoff?

When using **sp_configure,** you must always run either RECONFIGURE (or RECONFIGURE WITH OVERRIDE for the two indicated above) after setting a configuration option.

Example 1: The allow updates option requires RECONFIGURE WITH OVERRIDE.

SQL

```
EXEC sp_configure   'allow updates'
```

Result

name	minimum	maximum	config_value	run_value
allow updates	0	1	0	0

SQL

```
EXEC sp_configure   'allow updates'   ,  1
EXEC sp_configure   'allow updates'
```

Result

name	minimum	maximum	config_value	run_value
allow updates	0	1	1	0

SQL

```
RECONFIGURE WITH OVERRIDE
EXEC sp_configure   'allow updates'
```

Result

name	minimum	maximum	config_value	run_value
allow updates	0	1	1	1

Example 2: The 'fill factor' option also requires server stop and restart

SQL

```
EXEC sp_configure   'fill factor'
```

Result

name	minimum	maximum	config_value	run_value
fill factor (%)	0	100	0	0

SQL

```
EXEC sp_configure   'fill factor' , 80

EXEC sp_configure   'fill factor'
```

Result

name	minimum	maximum	config_value	run_value
fill factor (%)	0	100	80	0

Option fill factor requires reconfigure then server stop and restart so the config_value is shown as changed but not the run_value.

SQL

```
reconfigure
EXEC sp_configure   'fill factor'
```

Result

name	minimum	maximum	config_value	run_value
fill factor (%)	0	100	80	0

Still no change until we stop and restart the server, which we do now.

SQL

```
EXEC sp_configure   'fill factor' -- After server stop and restart
```

Result

name	minimum	maximum	config_value	run_value
fill factor (%)	0	100	80	80

2.8.1.9 RECONFIGURE

The reconfigure command updates the currently configured value of a configuration option changed with the **sp_configure** system stored procedure (the **config_value** column in the **sp_configure** result set). Some configuration options require a server stop and restart to update the currently running value. Therefore, **RECONFIGURE** does not always update the currently running value (the **run_value** column in the **sp_configure** result set) for a changed configuration value.

Syntax

```
RECONFIGURE [ WITH OVERRIDE ]
```

2.8.1.10 RECONFIGURE—Without the WITH OVERRIDE Option

The reconfigure command without the override option specifies that, if the configuration setting does not require a server stop and restart, the currently running value should be updated. Afterward the **config_value** and **run_value** should be the same for those options not requiring server stop and restart.

RECONFIGURE also checks the new configuration value for either invalid values or nonrecommended values.

2.8.1.11 RECONFIGURE WITH OVERRIDE

Without OVERRIDE, RECONFIGURE is for allow updates and recovery interval only. This allows invalid or nonrecommended values to be to be set for:

allow updates—default of 0 does not allow updates to system tables using DML (INSERT, UPDATE, DELETE). System procedures must be used. Setting to 1 is not recommended and requires WITH OVERRIDE.

recovery interval—default is 0 (self-configuring), recommended is 0 or 1 to 60. The value is the maximum number of minutes to recover each database. Over 60 minutes is not recommended and requires WITH OVERRIDE.

Books Online says the following.

> Keep **recovery interval** set at 0 (self-configuring) unless you notice that checkpoints are impairing performance because they are occurring too frequently. If this is the case, try increasing the value in small increments.[5]

Example:

SQL

```
EXEC sp_configure   'recovery interval'
```

Result

name	minimum	maximum	config_value	run_value
recovery interval (min)	0	32767	0	0

SQL

```
EXEC sp_configure   'recovery interval' , 120 -- 120 minutes = 2 hours
```

Result

```
DBCC execution completed. If DBCC printed error messages, contact your system administrator.
Configuration option 'recovery interval (min)' changed from 0 to 120.
Run the RECONFIGURE statement to install.
```

SQL

```
EXEC sp_configure   'recovery interval'
```

Result

name	minimum	maximum	config_value	run_value
recovery interval (min)	0	32767	0	0

5. Microsoft SQL Server 2K Books Online

SQL

```
PRINT 'RECONFIGURE'
RECONFIGURE
```

Result

```
RECONFIGURE
Server: Msg 5807, Level 16, State 1, Line 2
Recovery intervals above 60 minutes not recommended.
Use the RECONFIGURE WITH OVERRIDE statement to force this configuration.
```

SQL

```
EXEC sp_configure   'recovery interval' -- No change, RECONFIGURE is
                                           not strong enough
```

Result

name	minimum	maximum	config_value	run_value
recovery interval (min)	0	32767	120	0

SQL

```
PRINT  'RECONFIGURE WITH OVERRIDE'
RECONFIGURE WITH OVERRIDE
```

Result

```
RECONFIGURE WITH OVERRIDE
```

SQL

```
EXEC sp_configure   'recovery interval'
```

Result

name	minimum	maximum	config_value	run_value
recovery interval (min)	0	32767	120	120

2.8.1.12 SQL Server Settings in Enterprise Manager

Primary configuration settings for SQL Server are accessible in Enterprise Manager from the server properties dialog for a selected server as shown in the figure. In EM, right click on the server name and select "Properties" (see Figure 2-4).

Examine the settings available on these tabs and see Books Online for further details.

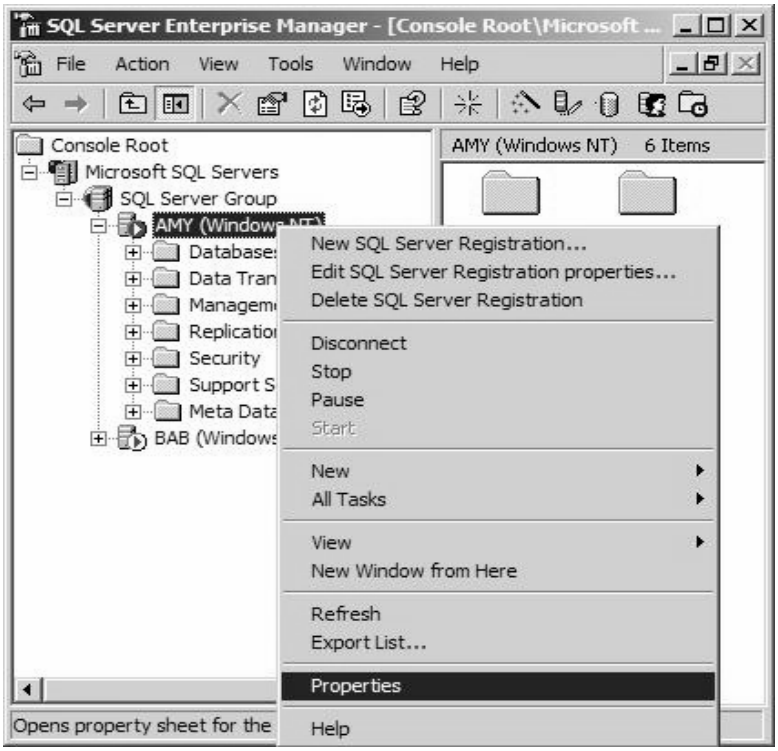

Figure 2-4 The Server Properties Dialog box for the AMY server.

2.8.2 Database Configuration (Database Properties)

Most database settings are best left as the default unless there is a specific reason to change. Database settings that you may need to a change occasionally are MULTI_USER and RESTRICTED_USER or SINGLE_USER in order to do maintenance. One may also want to set a specific database to READ_ONLY if its users do not need to change the data. A summary of database configuration statements is given in Figure 2-5. For more detail see Books Online, Index: database options.

Configuration settings for each database are assigned with

- **ALTER DATABASE** with a **SET** clause (all settings)

Configuration settings for each database are observed using

- **SELECT DATABASEPROPERTYEX(** ***'dbname'*** **,** ***'propertykeyword'*** **)**
  - The propertykeyword is not the same as the ALTER DATABASE keyword.
  - A complete list of **propertykeywords** is provided below in Table 2-78 (and in Books Online).
  - The **propertykeyword**, if there is one, is repeated in Table 2-77 in the "Description" column on a separate line and in parentheses.

Configuration settings for each database are **observed and assigned** with

- **Enterprise Manager** (primary settings only) - Right click on database name in EM
  - *ServerName* - Databases - *Databasename*

  and select **Properties**. When the dialog appears, click on the **Options** tab.
- **EXEC sp_dboption -obsolete**. Use ALTER DATABASE instead.

For more detail see Books Online, Index: database options.

Figure 2-5 Summary of Database Configuration Statements.

2.8.2.1 ALTER DATABASE dbname SET option

Only ALTER DATABASE with the SET clause, which may be used to change database settings, will be discussed in this section. See page 250 for the main coverage of ALTER DATABASE. See Table 2-77 for a summary of database configuration option keywords.

SQL Server 2K database options are set using ALTER DATABASE with a SET clause. In previous versions of SQL Server, database options were set with the **sp_dboption** system stored procedure. SQL Server 2K continues to support **sp_dboption**, which has been rewritten to call ALTER DATABASE, but it may not do so in the future.

DATABASEPROPERTYEX() may be used to show current settings for database options.

Partial Syntax

```
ALTER DATABASE  databasename  SET <optionspec>  [WITH
<termination>]
```

```
< optionspec >   See table
< termination > ::=
            ROLLBACK AFTER integer [ SECONDS ]
            | ROLLBACK IMMEDIATE
            | NO_WAIT
```

Table 2-77 ALTER Database Configuration Option Keywords

<optionspec> Keyword for both ALTER DATABASE and SET	Has Session Setting	Database Default	Description (DATABASEPROPERTYEX keyword)
Database State Options [abc]	[d]		See also Books Online: DATABASEPROPERTYEX and Setting Database Options
SINGLE_USER | RESTRICTED_USER | MULTI_USER	No	MULTI_ USER	Determines who may connect to the specified database. Example below. **(UserAccess)**
OFFLINE | ONLINE	No	ONLINE	When put OFFLINE the database is shutdown and can not be accessed. **(Status)**
READ_ONLY | READ_WRITE	No	READ_ WRITE	When put READ_ONLY users can not modify the database. **(Updateability)**
Cursor Options			
CURSOR_CLOSE_ON_COMMIT { ON | OFF }	Yes	OFF[b]	ON: (SQL-92) All open cursors are closed when a transaction is committed. OFF: Cursors must be closed explicitly and may cross transaction boundaries. **(IsCloseCursorsOnCommitEnabled)**
CURSOR_DEFAULT { LOCAL | GLOBAL }	No	GLOBAL	GLOBAL—cursors default to GLOBAL LOCAL—cursors default to LOCAL. Cursors may always be explicitly defined as LOCAL or GLOBAL. See Cursors page 638. **(IsLocalCursorsDefault)**

Table 2-77 ALTER Database Configuration Option Keywords (cont.)

<optionspec> Keyword for both ALTER DATABASE and SET	Has Session Setting	Database Default	Description (DATABASEPROPERTYEX keyword)
Automatic Options			
AUTO_CLOSE { ON | OFF } Comment from Books Online The AUTO_CLOSE option is useful for desktop databases because it allows database files to be managed as normal files. They can be moved, copied to make backups, or even e-mailed to other users. The AUTO_CLOSE option should not be used for databases accessed by an application that repeatedly makes and breaks connections to SQL Server. The overhead of closing and reopening the database between each connection will impair performance.	No	ON for SS 2000 Desktop Engine (MSDE 2000) OFF for all other SS 2000 editions	ON: the database is closed and shut down cleanly when the last user of the database exits and when all processes in the database are complete, thereby freeing any resources. The database reopens automatically when a user tries to use the database again. OFF: the database remains open even if no users are currently using it. **(IsAutoClose)**
AUTO_CREATE_STATISTICS { ON | OFF }	No	ON	ON: statistics are automatically created on columns without an index used in a predicate so as to speed the query. OFF: statistics not automatically created; but they can be manually created. **(IsAutoCreateStatistics)**
AUTO_UPDATE_STATISTICS { ON | OFF }	No	ON	ON: existing statistics are automatically updated when they become out-of-date. OFF: statistics are not automatically updated but can be manually updated. **(IsAutoUpdateStatistics)**
AUTO_SHRINK { ON | OFF } Default: ON for SS 2000 Desktop Engine (MSDE 2000) OFF for all other SS 2000 editions	No	See first column	ON: the database data and log files are periodically checked for unused space. OFF: files are not periodically checked for unused space. It is not possible to shrink a read-only database. **(IsAutoShrink)**

Table 2-77 ALTER Database Configuration Option Keywords (cont.)

<optionspec> Keyword for both ALTER DATABASE and SET	Has Session Setting	Database Default	Description (DATABASEPROPERTYEX keyword)
ANSI SQL-92 Compliance Options			
ANSI_NULL_DEFAULT { ON | OFF} This keyword applies to ALTER DATABASE only. The corresponding session SET keywords are ANSI_NULL_DFLT_ON and ANSI_NULL_DFLT_OFF	No	OFF but effectively ON[a]	Sets **Default Nullability** of a column—If ON is specified, CREATE TABLE follows SQL-92 rules to determine whether a column allows null values. OLE DB and ODBC set this to ON. **(IsAnsiNullDefault)**
ANSI_NULLS { ON | OFF }	Yes	OFF but effectively ON[a]	ON: SQL-92 behavior, comparing to NULL with = and <> returns NULL. OFF: NULL = NULL returns TRUE. **(IsAnsiNullsEnabled)**
ANSI_PADDING { ON | OFF }	Yes	OFF but effectively ON[a]	ON: Does NOT trim explicit trailing blanks in varchar and trailing zeros in varbinary columns. OFF: Does trim them. Books Online Recommendation: Leave t ON. **(IsAnsiPaddingEnabled)**
ANSI_WARNINGS { ON | OFF }	Yes	OFF but effectively ON[a]	ON means SQL-92 standard behavior of raising error messages or warnings for conditions like divide-by-zero and arithmetic overflow. **(IsAnsiWarningsEnabled)**
ARITHABORT { ON | OFF } Default: Query Analyzer sets ARITHABORT to ON for each session	Yes	OFF[c]	ON: Terminates a query if overflow or divide-by-zero occurs during query. OFF: Warning message displayed and processing continues. **(IsArithmeticAbortEnabled)**
Miscellaneous SET Options			
CONCAT_NULL_YIELDS_NULL	Yes	OFF but effectively ON[a]	ON: Concatenating NULL yields NULL (ON) versus empty string (OFF) **(IsNullConcat)**

Table 2-77 ALTER Database Configuration Option Keywords (cont.)

<optionspec> Keyword for both ALTER DATABASE and SET	Has Session Setting	Database Default	Description (DATABASEPROPERTYEX keyword)
NUMERIC_ROUNDABORT { ON | OFF }	Yes	OFF	ON: an error is generated when loss of precision occurs in an expression. OFF: the result is rounded to the precision of the destination with no error. **(IsNumericRoundAbortEnabled)**
QUOTED_IDENTIFIER { ON | OFF }	Yes	OFF but effectively ON[a]	See QUOTED_IDENTIFIER discussion with examples page 44. **(IsQuotedIdentifiersEnabled)**
RECURSIVE_TRIGGERS { ON | OFF }	No	OFF	ON allows triggers to fire recursively. **(IsRecursiveTriggersEnabled)**
Recovery Mode Statements			
RECOVERY { FULL | BULK_LOGGED | SIMPLE }	No	FULL -- except MSDE 2000 is SIMPLE	See Recovery Models page 559. **(Recovery)**
TORN_PAGE_DETECTION { ON | OFF } A torn page occurs when not all 16 sectors (512 bytes) of the 8 KB database page can be written to disk, as in power loss.	No	ON	ON causes the database to be marked as suspect if a torn page is found during recovery. If a torn page is found the database should be restored. This option should be left ON. **(IsTornPageDetectionEnabled)**

a. OLE DB and ODBC explicitly set this to ON for each client connection overriding database setting. See p. 215.

b. OLE DB and ODBC explicitly set this to OFF for each client connection overriding database setting. See p. 215.

c. Query Analyzer sets ARITHABORT to ON for each of its connections overriding database setting. See p. 216.

d. Database Options marked with "Yes" in the Session Setting column have corresponding session (connection) options which, if SET at the session level, take precedence over the database setting. See page 203.

2.8.2.2 Examples—ALTER DATABASE to Change UserAccess of a Database

Database Access Modes determines who may connect to the specified database as follows.

- MULTI_USER: Allows all users with database access privilege to connect
- RESTRICTED_USER: Allows only members of db_owner, dbcreator and sysadmin
- SINGLE_USER: Allows only the user issuing the ALTER DATABASE statement

Examples: You may read current access mode of the **pubs** database as shown.

```
SQL
SELECT DATABASEPROPERTYEX( 'pubs' , 'UserAccess' )

Result
----------------------------
MULTI_USER
```

Now set the access mode to any of the three values using ALTER DATABASE.

```
SQL ALTER DATABASE pubs  SET MULTI_USER
```

Setting to Either RESTRICTED_USER or SINGLE_USER Database Access
The following form waits indefinitely if unqualified users are connected to the database.

```
SQL ALTER DATABASE pubs  SET SINGLE_USER  -- may wait indefinitely
```

WITH NO_WAIT causes the ALTER DATABASE to fail immediately if unqualified users are connected to the specified database.

```
SQL
  ALTER DATABASE pubs  SET RESTRICTED_USER  WITH NO_WAIT
  ALTER DATABASE pubs  SET SINGLE_USER   WITH NO_WAIT
```

This command returns immediately. The new access can be seen with the following.

```
SQL
SELECT DATABASEPROPERTYEX( 'pubs' , 'UserAccess' )

Result
----------------------------
SINGLE_USER
```

WITH ROLLBACK IMMEDIATE forces immediate rollback of open transactions and terminates the connections of all unqualified users of the database.

```
SQL
ALTER DATABASE pubs SET RESTRICTED_USER  WITH ROLLBACK IMMEDIATE
ALTER DATABASE pubs SET SINGLE_USER  WITH ROLLBACK IMMEDIATE
```

WITH ROLLBACK AFTER *integer* **[SECONDS]** rolls back transactions and breaks the connections of all unqualified database users after the specified number of seconds.

```
SQL
ALTER DATABASE pubs
  SET RESTRICTED_USER  WITH ROLLBACK AFTER 60
ALTER DATABASE pubs
  SET SINGLE_USER  WITH ROLLBACK AFTER 60 SECONDS
```

Example: Set Recovery model for database mydb1 to FULL.

```
SQL ALTER DATABASE mydb1 SET RECOVERY FULL
```

Example: Set database mydb1 access to RESTRICTED_USER (allowing only members of **sysadmin** and **dbcreator** fixed server roles and **db_owner** fixed database roles). Unauthorized users currently connected will be unceremoniously disconnected and open transactions rolled back 60 seconds from the time the statement is executed.

```
SQL
ALTER DATABASE mydb1 SET RESTRICTED_USER
  WITH ROLLBACK AFTER 60 SECONDS
```

Change access for database mydb1 back to MULTI_USER.

```
SQL
ALTER DATABASE mydb1 SET MULTI_USER
  SELECT DATABASEPROPERTYEX( 'pubs' , 'UserAccess' )
```

SQL (cont.)

Result

```
---------------------------
MULTI_USER
```

The code **dbo** stands for database owner, the predefined user name in each database who is able to perform all database operations. Any **sysadmin** server role member becomes **dbo** inside each database.

2.8.2.3 DATABASEPROPERTYEX—Displays Database Settings

This function returns the current setting of the specified property in the specified database.

Syntax

```
SELECT DATABASEPROPERTYEX( 'dbname' , 'propertykeyword' )
```

Table 2-78 below lists all of the DATABASEPROPERTYEX property keywords, and some examples appear below the table. Most of these keywords were also listed above in the ALTER DATABASE keyword table (Table 2-77). A few examples were given there.

Additional database options are listed in Table 2-78.

Table 2-78 Keywords for DATABASEPROPERTYEX

DATABASE PROPERTYEX keyword	Description	Value Returned
Collation	Default collation name for the database.	Collation name
IsAnsiNullDefault	Database follows SQL-92 rules for allowing null values.	1=TRUE, 0=FALSE, NULL=Bad input
IsAnsiNullsEnabled	All comparisons to a null evaluate to null.	1=TRUE, 0=FALSE, NULL=Bad input
IsAnsiPaddingEnabled	Strings are padded to the same length before comparison or insert.	1=TRUE, 0=FALSE, NULL=Bad input
IsAnsiWarningsEnabled	Error or warning messages are issued when standard error conditions occur.	1=TRUE, 0=FALSE, NULL=Bad input
IsArithmeticAbortEnabled	Queries are terminated when an overflow or divide-by-zero error occurs.	1=TRUE, 0=FALSE, NULL=Bad input
IsAutoClose	Database shuts down cleanly and frees resources after the last user exits.	1=TRUE, 0=FALSE, NULL=Bad input

Table 2-78 Keywords for DATABASEPROPERTYEX (cont.)

DATABASE PROPERTYEX keyword	Description	Value Returned
IsAutoCreateStatistics	Existing statistics are automatically updated when they become out-of-date.	1=TRUE, 0=FALSE, NULL=Bad input
IsAutoShrink	Database files are candidates for automatic periodic shrinking.	1=TRUE, 0=FALSE, NULL=Bad input
IsAutoUpdateStatistics	Auto update statistics database option is enabled.	1=TRUE, 0=FALSE, NULL=Bad input
IsCloseCursorsOnCommitEnabled	Cursors that are open when a transaction is committed are closed.	1=TRUE, 0=FALSE, NULL=Bad input
IsFulltextEnabled	Database is full-text enabled.	1=TRUE, 0=FALSE, NULL=Bad input
IsInStandBy	Database is online as read-only, with restore log allowed.	1=TRUE, 0=FALSE, NULL=Bad input
IsLocalCursorsDefault	Cursor declarations default to LOCAL.	1=TRUE, 0=FALSE, NULL=Bad input
IsMergePublished	The tables of a database can be published for replication, if replication is installed.	1=TRUE, 0=FALSE, NULL=Bad input
IsNullConcat	Null concatenation operand yields NULL.	1=TRUE, 0=FALSE, NULL=Bad input
IsNumericRoundAbortEnabled	Errors are generated when loss of precision occurs in expressions.	1=TRUE, 0=FALSE, NULL=Bad input
IsQuotedIdentifiersEnabled	Double quotation marks can be used on identifiers.	1=TRUE, 0=FALSE, NULL=Bad input
IsRecursiveTriggersEnabled	Recursive firing of triggers is enabled.	1=TRUE, 0=FALSE, NULL=Bad input
IsSubscribed	Database can be subscribed for publication.	1=TRUE, 0=FALSE, NULL=Bad input
IsTornPageDetectionEnabled	SQL Server detects incomplete I/O operations caused by power failures, etc.	1=TRUE, 0=FALSE, NULL=Bad input
Recovery	Recovery model for the database.	FULL = full recovery model BULK_LOGGED = bulk logged model SIMPLE = simple recovery model
SQLSortOrder	SQL Server sort order ID supported in previous versions of SQL Server.	0 = Database uses Windows collation >0 = SQL Server sort order ID

Table 2-78 Keywords for DATABASEPROPERTYEX (cont.)

DATABASE PROPERTYEX keyword	Description	Value Returned
Status	Database status.	ONLINE = database is available OFFLINE = db was taken offline RESTORING = db is being restored RECOVERING = db is recovering and not yet ready for queries SUSPECT = db cannot be recovered
Updateability	Indicates whether data can be modified.	READ_ONLY READ_WRITE
UserAccess	Which users can access the database.	SINGLE_USER = only one user of **db_owner**, **dbcreator**, **sysadmin** RESTRICTED_USER = any of **db_owner**, **dbcreator**, **sysadmin** MULTI_USER = all users
Version	Database Version number for internal use only by SQL Server tools.	Integer = Database is open NULL = Database is closed

Example:

```
SQL
SELECT DATABASEPROPERTYEX(  'pubs' ,  'IsFulltextEnabled' )

Result
-----------
0
```

This says that full text searches are not presently enabled on the pubs database.

Many DATABASEPROPERTYEX keywords are also listed in the ALTER DATABASE table in the preceding section.

```
SQL
SELECT DATABASEPROPERTYEX(  'pubs' ,  'UserAccess' )
```

SQL (cont.)

Result

```
-----------------
MULTI_USER
```

2.8.2.4 Database Level Settings in Enterprise Manager

Primary configuration settings for SQL Server databases are accessible in Enterprise Manager from the server properties dialog for a selected server and database.

Expand the Console Tree in Enterprise Manager under the desired server.

Select your Server Name – Databases – <database name>

Right click on the <database name> and select Properties.

The tabs available for the database Properties dialog are:
General Data Files Transaction Log Filegroups Options Permissions

Options tab—The options tab, shown in Figure 2-6, has some settings that can be set from this tab or from the command line using ALTER DATABASE (see page 187).

Access

Restrict Access: db_owner, dbcreator, sysadmin only or Single user

Read-only

Recovery Model: Simple or Bulk-Logged or Full

Settings to allow or disallow features such as ANSI NULL default.

Compatibility Level: 60 or 65 or 70 or 80

2.8.2.5 `sp_dboption`—Brief Description as It Is Replaced by ALTER DATABASE

The stored procedure **sp_dboption** displays or changes database options. It is provided only for backward compatibility and might not appear in future releases of SQL Server. ALTER DATABASE is now recommended.

sp_dboption should not be used on either the master or tempdb databases.

Syntax

```
sp_dboption     [   [ @dbname = ] 'database' ]
                [ , [ @optname = ] 'option_name' ]
                [ , [ @optvalue = ] 'value' ]
```

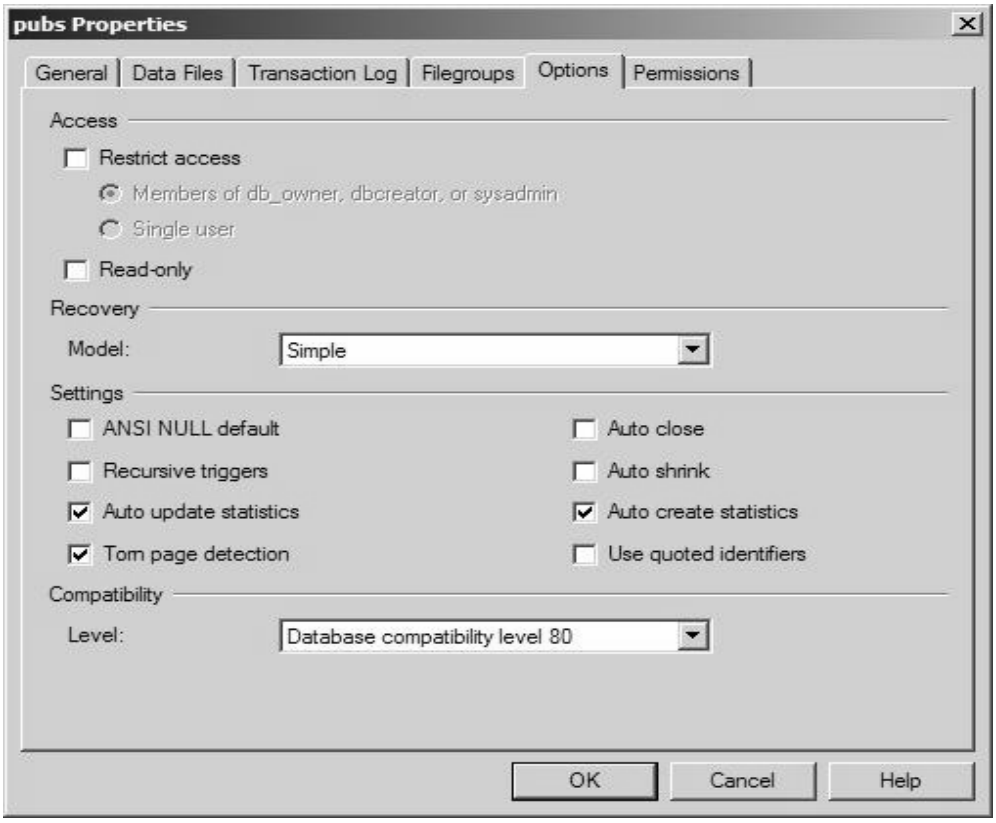

Figure 2-6 The Options Tab of the Properties Dialog Box.

These settings display or change global configuration settings for the current server.

sp_dboption may be executed with 0, 1 or 2 arguments as follows.

- 0 arguments: Lists all configuration setting names addressable with sp_dboption
- 1 argument: Displays the current settings that are set for the database specified
- 2 arguments: Displays the current setting of the specified option in the named database
- 3 arguments: Sets the specified option in the named database to the specified value

Examples:

SQL

```
EXEC sp_dboption
```

Result

```
Settable database options:
----------------------------------
ANSI null default
...
dbo use only
...
```

SQL

```
EXEC sp_dboption  pubs  -- shows pubs settings which are
"set"
```

Result

```
The following options are set:
----------------------------------
trunc. log on chkpt.
torn page detection
auto create statistics
auto update statistics
```

SQL

```
EXEC sp_dboption pubs , 'dbo use only' --  'dbo use only' is off'
```

Result

```
OptionName                       CurrentSetting
-------------------------------- --------------
dbo use only                     off
```

SQL

```
EXEC sp_dboption pubs , 'dbo use only', TRUE --  turn in on'
```

Result

```
The command(s) completed successfully.
```

2.8.2.6 Database Compatibility Level —`sp_dbcmptlevel`

MSS 2000 (version 8.0) implements SQL-92 more thoroughly than earlier versions, and it also adds new keywords. When upgrading a database from an earlier version of MSS, especially MSS 6.0 or 6.5, some of these changes may conflict with your existing application code.

Though running on SQL Server 2K, you may set a database to behave like an earlier version of SQL by using **sp_dbcmptlevel** system stored procedure. This will keep your production database operational while giving you a chance to rewrite your code. See Table 2-79.

Table 2-79 Compatibility Levels

Compatibility Level	Version
80	SQL Server 2K (version 8.0)
70	SQL Server 7.0
65	SQL Server 6.5
60	SQL Server 6.0

`sp_dbcmptlevel` sets the specified database to behave according to the specified version of SQL Server.

Syntax

```
sp_dbcmptlevel    [ [ @dbname = ] name ]     [ , [ @new_cmptlevel = ] version ]
```

Argument

version

The version of SQL Server with which the database is to be made compatible. The value must be 80, 70, 65 or 60.

References

Books Online: **sp_dbcmptlevel;** compatibility issues, overview

2.8.3 Session (Connection) Configuration Settings

Session or connection settings are values that apply to the current connection of a client program for the SQL Server database engine. They will remain in effect until the end of the session (when the connection is closed) or a SET statement is issued as described below.

*S*ession settings that can be quite useful for debugging or performance testing include NOEXEC, NOCOUNT, PARSEONLY, SHOWPLAN_xxx, STATISTICS xxx, etc. These options are well worth studying and testing to see the information they provide.

To determine how your current session settings are determined, I suggest starting with the OLE DB and ODBC driver connection settings (see page 215) and Query Analyzer connection settings (see page 216) and then reading the section on Which Session Setting Is in Effect? (see page 219).

Some session settings will be changed by the client libraries (OLE DB and ODBC) and by Query Analyzer, if that is the client program. And all session option settings may be changed by the user using the SET command.

Figure 2-7 is a summary of session configuration statements. The pages that follow contain detailed information about the information in the box.

2.8.3.1 sp_configure user options

sp_configure is a server setting that affects future sessions. The options allow a user to set all 15 default session query processing options applicable ONLY FOR NEW LOGIN SESSIONS (CONNECTIONS). Anyone currently logged in is not affected until the next time they log in.

The **sp_configure** USER OPTIONS value is a single integer representing a bitset specifying global defaults for 15 settings that affect each user's session (connection). A user may override any setting by changing it with the SET statement.

Executing **sp_configure 'user options'**, *value* assigns default settings for new logins.

Any user may use SET to override any setting for the current session.

If a user has SET an option then that setting is used for the current session Recall ODBC and OLE DB set some options when connecting or if the current database has a setting for the option then it will be used or if **sp_configure 'user options'** for the option is in effect it will be used or the default setting for the option will be used.

The options settable by **sp_configure 'user options'** and the setting value are the same as those visible with **@@OPTIONS** (page 208) and are listed in Table 2-80.

New sessions inherit server or database options and may change some of them. Configuration settings for the current session or connection are changed using

- **sp_configure 'user options'**
  Assigns default session settings, which are in effect unless overridden by a SET command. See full description page 204.
  Client libraries often issue SET statements for each connection. See OLE DB and ODBC driver connection settings, page 215.
  Query Analyzer additionally assigns SET options, which may be configured by the user. See Query Analyzer connection settings, page 216.
- **SET** (page 204)

Configuration settings for each session or connection are observed using

- **@@OPTIONS** — Displays integer bitset of set options (page 209)
- **DBCC USEROPTIONS**—Displays all options currently set (page 210)
- **SELECT SESSIONPROPERTY ('*option*')** —Displays an option setting (page 211)
- **Session Configuration Functions** —Each displays an option setting (page 214)

Primary settings for each Query Analyzer connection are observed and changed using

- Query Analyzer primary session settings.

See also Books Online: SET Options That Affect Results.

Figure 2-7 Session Configuration Statements Summary.

Syntax

```
sp_configure 'user options'      [ , [ @configvalue = ]  value ]
```

value = The sum of the values of all options desired to be set for future new logins.

Remember to run **RECONFIGURE** to make the change effective.

Table 2-80 sp_configure USER OPTIONS

Value	Option	Description — Behavior when ON
1	DISABLE_DEF_CNST_CHK	Controls interim or deferred constraint checking.
2	IMPLICIT_TRANSACTIONS	Controls whether a transaction is committed automatically (OFF) when a statement is executed or the transaction requires explicit commit (ON).
4	CURSOR_CLOSE_ON_COMMIT	Controls behavior of cursors after a commit operation has been performed.
8	ANSI_WARNINGS	Controls truncation and NULL in aggregate warnings.

Table 2-80 sp_configure USER OPTIONS (cont.)

Value	Option	Description — Behavior when ON
16	ANSI_PADDING	Controls padding of character variables. See page 102.
32	ANSI_NULLS	Controls NULL handling when using equality operators.
64	ARITHABORT	Terminates a query when an overflow or divide-by-zero error occurs.
128	ARITHIGNORE	Returns NULL when overflow or divide-by-zero error occurs during a query.
256	QUOTED_IDENTIFIER	Differentiates between single and double quotation marks when evaluating an expression.
512	NOCOUNT	Turns off the “how many rows affected” message at the end of each statement.
1024	ANSI_NULL_DFLT_ON	Alters the session's behavior to use ANSI compatibility for nullability. New columns defined without explicit nullability are defined to allow nulls.
2048	ANSI_NULL_DFLT_OFF	Alters the session to not use ANSI compatibility for nullability. New columns defined without explicit nullability will not allow nulls.
4096	CONCAT_NULL_YIELDS_NULL	Returns NULL when concatenating a NULL value with a string.
8192	NUMERIC_ROUNDABORT	Generates an error when a loss of precision occurs in an expression.
16384	XACT_ABORT	Rolls back a transaction if a Transact- SQL statement raises a run-time error.

For an example of the use of **sp_configure 'user options'**, see page 219.

2.8.3.2 SET

The SET statement assigns current session (connection) option settings. These settings are listed in Table 2-81.

Table 2-81 SET Statement Options

SET Command Option Keyword	Default Setting	@@OPTIONS value[d] See p. 209	Description
Date and Time Options	[a,b]		
DATEFIRST { 1|2|3|4|5|6|7 } 1=Monday, 7=Sunday	7 (Sunday)		Sets first day of week. **Ex:** SET DATEFIRST 7

Table 2-81 SET Statement Options (cont.)

SET Command Option Keyword	Default Setting	@@ OPTIONS value[d] See p. 209	Description
DATEFORMAT { mdy|dmy|ymd|ydm|myd|dym }	mdy		Sets the order of (month/day/year) for entering datetime or smalldatetime data. **Ex:** SET DATEFORMAT mdy
Locking Options			
DEADLOCK_PRIORITY { LOW | NORMAL | @*deadlock_var* }	NORMAL		Controls how session reacts if in deadlock. LOW — Current session is victim NORMAL — Let SQL Server decide @*deadlock_var* - 3=LOW, 6=NORMAL
LOCK_TIMEOUT *millisec_til_timeout*	1		Specifies the number of milliseconds a statement waits for a lock to be released.
Miscellaneous SET Options			See also Books Online: "SET Options"
CONCAT_NULL_YIELDS_NULL	OFFa	4096	ON means concatenating with NULL yields NULL versus empty string (OFF) OLE DB and ODBC set this to ON when making a new connection.
DISABLE_DEF_CNST_CHK	OFF	1	For backward compatibility only
FIPS_FLAGGER { ENTRY|FULL | INTERMEDIATE | OFF }			Specifies checking for compliance with the FIPS 127-2 standard, and specifies SQL-92 Entry, Full or Intermediate Level or None.
IDENTITY_INSERT	OFF		ON allows explicit values to be inserted into an identity column.

LANGUAGE { [N] '*language*' | @*language_var* }	us_english See p. 178.		Specifies the session language including **datetime** formats and system messages. **EXEC sp_helplanguage —** list languages
			
			Example:
			`SET LANGUAGE Deutsch PRINT CAST ( '2003-05-10 14:35' As DATETIME )`
			Okt 5 2003 2:35PM
			`SET LANGUAGE us_english PRINT CAST ( '2003-05-10 14:35' As DATETIME )`
			May 10 2003 2:35PM

OFFSETS *keyword_list*			Use only in DB-Library applications. See Books Online.

Table 2-81 SET Statement Options (cont.)

SET Command Option Keyword	Default Setting	@@ OPTIONS value[d] See p. 209	Description
Query Execution Statements			
ARITHABORT Note on Default: See footnote c.	OFF[c]	64	Terminates a query if overflow or divide-by-zero occurs during query.
ARITHIGNORE	OFF	128	ON means Error Message is returned from overflow or divide-by-zero.
FMTONLY	OFF		Returns only meta data, no data
NOCOUNT	OFF	512	Stops the message with number of rows affected from being returned.
NOEXEC	OFF		Parse and compile but do not execute.
NUMERIC_ROUNDABORT	OFF	8192	Sets level of error reporting when rounding causes a loss of precision.
PARSEONLY	OFF		Parse but do not execute from now on.
QUERY_GOVERNOR_COST_LIMIT *integervalue*	0 (unlimited)		**sysadmin** setting to disallow queries whose estimated run time exceeds the specified number of seconds. Default is 0, unlimited time, so all queries run.
ROWCOUNT *integervalue*	0 (unlim)		Stops processing the query after the specified number of rows.
TEXTSIZE *integervalue*	4 KB		Specifies the size in bytes of **text** and **ntext** data returned from a SELECT Either 0 or 4096 sets to default of 4 KB.
SQL-92 Settings Statements			
ANSI_DEFAULTS	n/a		ON sets all options in this section to ON except ANSI_NULL_DFLT_OFF to OFF. OFF leaves ANSI_NULL_DFLT_OFF unchanged and sets rest to OFF
ANSI_NULLS	OFF[a]	32	Sets ANSI SQL-92 compliant behavior in effect when comparing to NULL with equals (=) and not equal to (<>) .
ANSI_NULL_DFLT_ON	OFF[a]	1024	Only one of these two can be ON at a time. So setting one ON sets the other OFF. Both may be set to OFF at the same time.

Table 2-81 SET Statement Options (cont.)

SET Command Option Keyword	Default Setting	@@ OPTIONS value[d] See p. 209	Description
ANSI_NULL_DFLT_OFF	OFF	2048	
ANSI_PADDING	ON[a]	16	Set blank padding for values shorter than the defined size of the column and for values that have trailing blanks in char and binary data.
ANSI_WARNINGS	OFF[a]	8	ON means SQL-92 standard behavior of raising error messages or warnings for conditions like divide-by-zero and arithmetic overflow.
CURSOR_CLOSE_ON_COMMIT	OFF[b]		As described by the name when ON
QUOTED_IDENTIFIER	OFF[a]	256	See QUOTED_IDENTIFIER discussion with examples page 44.
IMPLICIT_TRANSACTIONS	OFF[b]	2	See details with **Transactions** below.
Statistics Statements			
FORCEPLAN	OFF		Makes the query optimizer process a join in the same order as tables appear in the FROM clause of a SELECT statement.
SHOWPLAN_ALL	OFF		ON: does not execute SQL statements but instead returns the detailed execution plan and estimates of the resource requirements to execute the statements.
SHOWPLAN_TEXT	OFF		ON: does not execute SQL statements but instead returns the execution plan for the statements.
STATISTICS IO	OFF		ON: displays the disk activity generated by Transact-SQL statements when executed.
STATISTICS PROFILE	OFF		ON: Displays profile information for a statement including number of rows produced and number of times the query ran.
STATISTICS TIME	OFF		Displays the time in milliseconds to parse, compile and execute each statement.

Table 2-81 SET Statement Options (cont.)

SET Command Option Keyword	Default Setting	@@ OPTIONS value[d] See p. 209	Description
Transaction Statements		[d]	See "Transaction Control," page 529.
IMPLICIT_TRANSACTIONS	OFFb	2	IMPLICIT_TRANSACTION mode ON requires an explicit COMMIT/ROLL BACK for each transaction. OLE DB and ODBC set this to OFF when making a new connection. When OFF, AUTOCOMMIT MODE is in effect. See Transaction Control, page 529.
REMOTE_PROC_TRANSACTIONS	OFF		Specifies that when a local transaction is active, executing a remote stored procedure starts a Transact-SQL distributed transac tion managed by the Microsoft Distributed Transaction Manager (MS DTC).
TRANSACTION ISOLATION LEVEL { READ UNCOMMITTED | READ COMMITTED | REPEAT-ABLE READ | SERIALIZABLE }	READ COM-MITTED		Controls the default locking behavior for the session (connection). See "Transaction Control," p. 529.
XACT_ABORT	OFF	16384	ON: rolls back the entire transaction if a statement raises a run-time error OFF: rolls back just the statement and the transaction continues.

a. OLE DB and ODBC explicitly set this to ON for each client connection overriding database setting. See p. 215.

b. OLE DB and ODBC explicitly set this to OFF for each client connection overriding database setting. See p. 215.

c. Query Analyzer sets ARITHABORT to ON for each of its connections overriding database setting. See p. 216.

d. See @@OPTIONS, p. 208.

2.8.3.3 @@OPTIONS

The value @@OPTIONS returns a bitmask of session options from Table 2-82 SET for the current connection. The value includes all options currently SET by virtue of server settings including **sp_configure 'user options'** and SET operations including those set by OLE DB and ODBC drivers (see page 215).

Bit positions in @@OPTIONS are identical to those in sp_configure 'user options' but the @@OPTIONS value represents current session settings of the options.

@@OPTIONS reports on the following 15 settings which includes the 7 options that SESSIONPROPERTY() reports. So @@OPTIONS is more complete.

Table 2-82 @@OPTIONS Settings

Option	Default [a,b]	@@OPTIONS Value
DISABLE_DEF_CNST_CHK	OFF	1
IMPLICIT_TRANSACTIONS	OFF[b]	2
CURSOR_CLOSE_ON_COMMIT	OFF[b]	4
ANSI_WARNINGS	OFF[a]	8
ANSI_PADDING	ON[a]	16
ANSI_NULLS	OFF[a]	32
ARITHABORT	OFF[c]	64
ARITHIGNORE	OFF	128
QUOTED_IDENTIFIER	OFF[a]	256
NOCOUNT	OFF	512
ANSI_NULL_DFLT_ON	OFF[a]	1024
ANSI_NULL_DFLT_OFF	OFF	2048
CONCAT_NULL_YIELDS_NULL	ON[a]	4096
NUMERIC_ROUNDABORT	OFF	8192
XACT_ABORT	OFF	16384

a. OLE DB and ODBC explicitly set this to ON for each client connection overriding database setting. See p. 215.

b. OLE DB and ODBC explicitly set this to OFF for each client connection overriding database setting. See p. 215.

c. Query Analyzer sets ARITHABORT to ON for each of its connections overriding database setting.. See p. 216.

See more examples displaying current session (connection) settings on page 220.

SQL

```
SELECT @@OPTIONS & 4096 -- Shows that CONCAT_NULL_YIELDS_NULL is currently ON
```

Result

```
4096
```

```
SQL
SELECT @@OPTIONS -- Shows the integer bitmask which
  includes all @@OPTIONS currently ON

Result
5496
```

```
SQL
SET CONCAT_NULL_YIELDS_NULL   OFF
SELECT @@OPTIONS & 4096      -- Shows that
CONCAT_NULL_YIELDS_NULL is currently OFF

Result
0
```

```
SQL
SELECT @@OPTIONS -- Shows the integer bitmask which
  includes all @@OPTIONS currently ON

Result
1400
```

2.8.3.4 DBCC USEROPTIONS

DBCC USEROPTIONS returns all SET options which are active (set) for the current session (connection).

Syntax

```
DBCC USEROPTIONS
```

Example: Example of ways to display current session (connection) settings.

```
SQL
DBCC USEROPTIONS

```

SQL (cont.)

Result

```
Set Option                          Value
----------------------------------- --------------------------
textsize                            64512
language
us_english
dateformat                          mdy
datefirst                           7
quoted_identifier                   SET
ansi_null_dflt_on                   SET
ansi_defaults                       SET
ansi_warnings                       SET
ansi_padding                        SET
ansi_nulls                          SET
concat_null_yields_null             SET
DBCC execution completed. If DBCC printed error messages, contact your system administrator.
```

2.8.3.5 SESSIONPROPERTY

SESSIONPROPERTY returns the current setting of one of the seven session options listed in Table 2-83. Returns on the setting are listed in Table 2-84.

Returns 1 if SET, 0 if NOT SET and NULL if the input *option* name was invalid.

Syntax

```
SESSIONPROPERTY ( 'option' )
```

Arugment

option

The SESSIONPROPERTY *option* names are the same as for ALTER DATABASE.

Table 2-83 SESSIONPROPERTY Options

Option Name	Option Name
ANSI_NULLS	CONCAT_NULL_YIELDS_NULL
ANSI_PADDING	NUMERIC_ROUNDABORT
ANSI_WARNINGS	QUOTED_IDENTIFIER
ARITHABORT	

For the meaning of each option see ANSI SQL-92 Compliance Options, see page 191.

Table 2-84 Returns

Return Value	Option Is Currently
1	ON
O	OFF
NULL	Invalid Option name

Examples using SESSIONPROPERTY()

```
SQL
SELECT  SESSIONPROPERTY( 'QUOTED_IDENTIFIER' )   -- Option ON returns 1

Result
--------
1
```

```
SQL
SELECT  SESSIONPROPERTY( 'NUMERIC_ROUNDABORT' )   -- Option OFF returns 0

Result
--------
0
```

```
SQL
SELECT  SESSIONPROPERTY( 'Foo_Foo' ) -- Invalid input option name, returns
  NULL

Result
--------
NULL
```

2.8.3.6 Comparing @@OPTIONS, DBCC USEROPTIONS and SESSIONPROPERTY()

The following methods show current session settings as indicated:

@@OPTIONS enables you to determine the setting of a specific option but it requires looking up the option number of interest and doing a bitwise AND to determine if a specific setting is on or off. Only the settings that have a value in the @@OPTIONS column of Table 2-81**,** page 204, may be read with this function.

DBCC USEROPTIONS is convenient since it reports all options that are currently set. It is silent on options not currently set.

SESSIONPROPERTY() returns the one option setting specified, as does @@OPTIONS, and it uses the same option keyword as SET, so it's more consistent in its use. But it is less complete than @@OPTIONS because it only reports on the seven options listed in Table 2-83.

Examples Comparing the Three:

SQL

```
SELECT @@OPTIONS & 1024 -- Shows that ANSI_NULL_DFLT_ON   is currently ON
```

Result

```
--------
1024
```

SQL

```
SELECT @@OPTIONS & 4096 -- Shows that CONCAT_NULL_YIELDS_NULL
  is currently ON
```

Result

```
--------
4096
```

SQL

```
SELECT  SESSIONPROPERTY( 'ANSI_NULL_DFLT_ON' ) -- Can't check this one
```

Result

```
--------
NULL
```

SQL

```
SELECT  SESSIONPROPERTY( 'CONCAT_NULL_YIELDS_NULL' )   -- Option ON returns 1
```

Result

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

SQL

```
DBCC USEROPTIONS
```

Result

```
Set Option                          Value
----------------------------------- --------------------------
ansi_null_dflt_on                   SET
...
concat_null_yields_null             SET
DBCC execution completed. If DBCC printed error messages, contact your system
administrator.
```

2.8.3.7 Session Configuration Functions

These built-in scalar functions return the current session setting indicated by the name. Table 2-85 provides a description.

Table 2-85 Session Configuration Functions

Function Name	Description
@@DATEFIRST	Returns the current value of the SET DATEFIRST parameter, which indicates the specified first day of each week: 1 for Monday, 2 for Wednesday, and so on through 7 for Sunday.
@@DBTS	Returns the value of the current **timestamp** data type for the current database. This **timestamp** is guaranteed to be unique in the database.
@@LANGID	Returns the local language identifier (ID) of the language currently in use.
@@LANGUAGE	Returns the name of the language currently in use.
@@LOCK_TIMEOUT	Returns the current lock time-out setting, in milliseconds, for the current session.

Table 2-85 Session Configuration Functions (cont.)

Function Name	Description
@@MAX_CONNECTIONS	Returns the maximum number of simultaneous user connections allowed on a Microsoft SQL Server. The number returned is not necessarily the number currently configured.
@@MAX_PRECISION	Returns the precision level used by **decimal** and **numeric** data types as currently set in the server.
@@NESTLEVEL	Returns the nesting level of the current stored procedure execution (initially 0).
@@OPTIONS	Returns information about current SET options.
@@REMSERVER	Returns the name of the remote Microsoft SQL Server database server as it appears in the login record.
@@SERVERNAME	Returns the name of the local server running Microsoft SQL Server.
@@SERVICENAME	Returns the name of the registry key under which Microsoft SQL Server is running. @@SERVICENAME returns MSSQLServer if the current instance is the default instance; this function returns the instance name if the current instance is a named instance.
@@SPID	Returns the server process identifier (ID) of the current user session.
@@TEXTSIZE	Returns the current value of the TEXTSIZE option of the SET statement, which specifies the maximum length, in bytes, of **text** or **image** data that a SELECT statement returns.
@@VERSION	Returns the date, version and processor type for the current installation of Microsoft SQL Server.

Example:

```
SQL
SELECT  @@SPID    -- Returns the id SQL Server has
  assigned the current session (connection)

Result
51
```

2.8.3.8 OLE DB and ODBC Driver Connection Settings

OLE DB and ODBC drivers make the following settings for every new connection.

ON

```
CONCAT_NULL_YIELDS_NULL
ANSI_NULL_DEFAULT
ANSI_DEFAULTS  -- which set all of the following to ON
  ANSI_NULLS BOL -- 'SET Options' for ANSI_DEFAULTS

  ANSI_NULL_DFLT_ON -- sets ANSI_NULL_DFLT_OFF to OFF
  ANSI_PADDING         -- See page 112.
  ANSI_WARNINGS
  QUOTED_IDENTIFIER
```

OFF

```
  CURSOR_CLOSE_ON_COMMIT
    IMPLICIT_TRANSACTIONS
```

See each item under "SETTING DATABASE OPTIONS" "SET CONCAT_NULL_YIELDS_NULL" for ODBC/OLE DB sessions settings.

ODBC and OLE DB first turn on the above settings identified as ON. Then they turn off the two items identified as OFF (they were set to ON when ANSI_DEFAULTS was set ON). See "SET ANSI_DEFAULTS" for ODBC/OLE DB sessions settings. These settings will be in effect for every ODBC and OLE DB client unless you change them with an explicit SET statement. See Query Analyzer additions next.

2.8.3.9 Query Analyzer Connection Settings

Query Analyzer uses ODBC, so it starts with the ODBC settings listed above in effect, then it sets the following additional options as shown.

ON

```
  ARITHABORT
```

OFF

```
  NOCOUNT
    NOEXEC
    PARSEONLY
    SHOWPLAN_TEXT
    STATISTICS TIME
    STATISTICS IO
```

0 `ROWCOUNT (0 or NULL means unlimited rows in result sets)`

The net result of these default actions can be confirmed by executing this statement in Query Analyzer.

```
SQL
-- In a Query Analyzer with default settings
  DBCC USEROPTIONS

Result
Set Option                     Value
------------------------------ -------------
textsize                       64512
language                       us_english
dateformat                     mdy
datefirst                      7
quoted_identifier              SET
arithabort                     SET
ansi_null_dflt_on              SET
ansi_defaults                  SET
ansi_warnings                  SET
ansi_padding                   SET
ansi_nulls                     SET
concat_null_yields_null        SET
```

These settings, except the first four, are set explicitly by ODBC and Query Analyzer as just described. The first four were inherited from the defaults as summarized in the next section. See also Books Online: Using SET Options in SQL Query Analyzer.

2.8.3.10 Changing Query Analyzer Default Connection Settings

You may change the default connection settings for your own Query Analyzer from Query—Current Connection Properties, which opens the dialog shown in Figure 2-8. Check a box for ON or uncheck for OFF and click Apply. Table 2-86 lists the default SET session settings made by Query Analyzer.

Table 2-86 Summary of All Default SET Session Settings Made by Query Analyzer

Option	Setting
Set nocount	OFF
Set noexec	OFF
Set parseonly	OFF

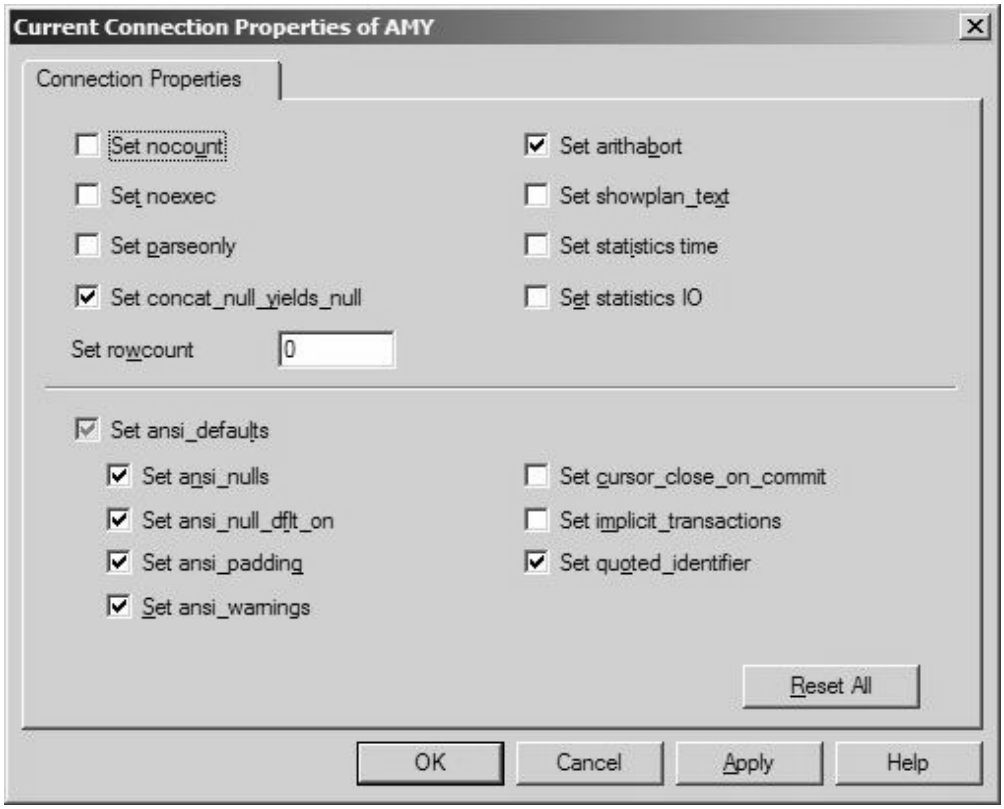

Figure 2-8 The Connection Properties Dialog Box in Query Analyzer.

Table 2-86 Summary of All Default SET Session Settings Made by Query Analyzer (cont.)

Option	Setting
Set concat_null_yields_null	ON
Set rowcount	0
Set ansi_defaults	ON
Set arithabort	ON
Set showplan_text	OFF
Set statistics time	OFF
Set statistics 10	OFF

Table 2-86 Summary of All Default SET Session Settings Made by Query Analyzer (cont.)

Option	Setting
Set ansi_nulls	ON
Set ansi_null_dflt_on	ON
Set ansi_padding	ON
Set ansi_warnings	ON
Set cursor_close_on_commit	OFF
Set implicit_transactions	OFF
Set quoted_identifier	ON

2.8.3.11 Which Session Setting Is in Effect?

See "Which Setting Is the One in Effect?" on page 184. What follows is my version of SQL Server's algorithm to decide which setting to use.

If a user has SET an option then that setting is used for the current session else if it is an option set by ODBC, OLE DB or Query Analyzer then it will be used (page 215) else if **sp_configure 'user options'** for the option is in effect it will be used (page 202) else if the current database has a setting for the option then it will be used (page 187) else the SQL Server default setting for the option will be used (page 203).

The case numbers are given below to identify what is happening in the following examples.

1. An explicit session SET statement takes precedence and lasts until changed by a new SET statement or the end of the session (connection).

2a. OLE DB and ODBC drivers make the following settings for each new connection.

ON

```
CONCAT_NULL_YIELDS_NULL
   ANSI_NULL_DEFAULT
   ANSI_DEFAULTS (which set esach of the following to ON)
      ANSI_NULLS
      ANSI_NULL_DFLT_ON (which sets ANSI_NULL_DFLT_OFF to OFF)
      ANSI_PADDING (see page 102)
      ANSI_WARNINGS
      CURSOR_CLOSE_ON_COMMIT
      IMPLICIT_TRANSACTIONS
      QUOTED_IDENTIFIER
```

OFF

CURSOR_CLOSE_ON_COMMIT
IMPLICIT_TRANSACTIONS

So these settings will be in effect for every ODBC and OLE DB client until you change them with an explicit SET statement. This is true regardless of **sp_configure 'user options'** or database options assigned with **ALTER DATABASE**.

2b. Query Analyzer sets ARITHABORT to ON (see page 216).

3. In the absence of 1 or 2, any option set with **sp_configure 'user options'** will be used.
4. **Database default** is next (can be set with **ALTER DATABASE**, pages 189 and 250).
5. Lastly, the **SQL Server default** will be used, page 202.

Examples Showing which Session Setting Is in Effect Examples are given here to demonstrate Case 1, 2a, 2b, 3 and 4. Each example starts with a new connection and shows user actions, if any, to change a setting and the result.

Example: **CASE 1**—Explicit SET CONCAT_NULL_YIELDS_NULL to OFF
Open a new database connection to the pubs database.

SQL

```
SELECT SESSIONPROPERTY( 'CONCAT_NULL_YIELDS_NULL' ) -- Show it's ON
```

Result

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

SQL

```
SET  COmAT_NULL_YIELDS_NULL  OFF             -- changes OLE DB setting

SELECT SESSIONPROPERTY( 'CONCAT_NULL_YIELDS_NULL' ) -- Show it's now OFF
```

Result

```
SQL
SELECT DATABASEPROPERTYEX( 'pubs' , 'IsNullConcat' )
  -- Show DB default is OFF

Result
-------
0
```

Table 2-87 contains a summary of actions in order of precedence. The first "Yes" from the left takes precedence.

Table 2-87 Case 1 Explicit SET of CONCAT_NULL_YIELDS_NULL Option

Explicit SET	Set by OLE DB/ODBC	User Option	Database Default
Yes**—SET to OFF**	Yes—ON but overridden	Would be overrid- den even if set	OFF but overridden

Example: CASE 2a—OLE DB sets CONCAT_NULL_YIELDS_NULL to ON. Open a new database connection to the pubs database.

```
SQL
 SELECT SESSIONPROPERTY( 'CONCAT_NULL_YIELDS_NULL' )
   -- Session setting is ON
 -- (I'm using Query Analyzer and OLE DB set it ON)

Result
-------
1
```

```
SQL
 SELECT  @@OPTIONS & 4096   -- Same result, this is an
   alternative to SESSIONPROPERTY
```

```
SQ (cont.)

Result

-------
4096
```

```
SQL

SELECT DATABASEPROPERTYEX( 'pubs' , 'IsNullConcat' )
  -- Show DB default is OFF


Result

-------
0
```

Table 2-88 contains a summary of actions in order of precedence. The first "Yes" from the left takes precedence.

Table 2-88 Case 2a—ODBC Set of CONCAT_NULL_YIELDS_NULL option

Explicit SET	Set by OLE DB/ ODBC	User Option	Database Default
No	Yes — ON	Would be overridden even if set	OFF but overridden

Example: CASE 2b—Query Analyzer sets ARITHABORT to ON.

See "Query Analyzer Connection Settings" on page 216.

Example: CASE 3—NUMERIC_ROUNDABORT, we'll change User Option to ON.

NUMERIC_ROUNDABORT is one of the few options not set by OLE DB or ODBC, so setting the default user option will have a visible effect. In Session 1 below we first demonstrate that no user options settings are in effect and that NUMERIC_ROUNDABORT defaults to OFF. Then we use sp_configure to set the new user default to ON. Session 1 won't be affected, so we open a new connection as Session 2 and see the new setting is ON.

Session 1 This session observes and changes the sp_configure 'user options', but only new login sessions will see the effect. Open a new database connection to the pubs database.

SQL

```
EXEC sp_configure  'user options' -- Show that no 'user
                                  options' are currently set

-- (run value is 0)
```

Result

name	minimum	maximum	config_value	run_value
user options	0	32767	0	0

SQL

```
-- For fun, show that the setting in this session is off before and after
the 'user option' is changed
SELECT SESSIONPROPERTY( 'NUMERIC_ROUNDABORT' ) -- Session setting is OFF
```

Result

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

Change the 'user option':

SQL

```
-- Set option for NUMERIC_ROUNDABORT to ON
EXEC sp_configure 'user options' , 8192

RECONFIGURE -- Don't forget that reconfigure is required
  to make the change effective

EXEC sp_configure 'user options'
  -- NUMERIC_ROUNDABORT 'user options is set' (8192)
```

Result

name	minimum	maximum	config_value	run_value
user options	0	32767	8192	8192

```
SQL
-- The setting is on for new sessions, but our
  NUMERIC_ROUNDABORT option setting is still OFF.
SELECT SESSIONPROPERTY( 'NUMERIC_ROUNDABORT' )

Result
-------
0
```

Session 2 New Login Sessions (Connections) will see the Change Open NEW Query Analyzer CONNECTION. The session setting is now ON.

```
SQL
Again show that the Database default for IsNumericRoundAbortEnabled = OFF
SELECT DATABASEPROPERTYEX( 'pubs' , 'IsNumericRoundAbortEnabled')

Result
-------
1
```

```
SQL
SELECT  @@OPTIONS & 8192 -- Same result as SESSIONPROPERTY

Result
-------
8192
```

```
SQL
 -- Show that the Database default for IsNumericRoundAbortEnabled = OFF
 SELECT DATABASEPROPERTYEX( 'pubs' , 'IsNumericRoundAbortEnabled')

Result
-------
0
```

```
-- Clean up by returning User Options to 0 for future
sessions
EXEC sp_configure  'user options' , 0
RECONFIGURE                          -- Don't forget
that reconfigure is required to make the change
effective
```

Summary of actions in order of precedence: The first "Yes" from the left takes precedence.

Table 2-89 Case 3—NUMERIC_ROUNDABORT Option— Recall 'user option' Affects Only New Sessions

Explicit SET	Set by OLE DB/ ODBC	User Option	Database Default
No	No	ON	OFF

Example: CASE 4—NUMERIC_ROUNDABORT option, uses the Database default. Open a new database connection to the pubs database.

SQL

```
SELECT DATABASEPROPERTYEX( 'pubs' ,
  'IsNumericRoundAbortEnabled') -- Default OFF
```

Result

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

SQL

```
SELECT SESSIONPROPERTY( 'NUMERIC_ROUNDABORT' )
  --Show session setting is also off
```

Result

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

SQL

```
SELECT  @@OPTIONS & 8192              -- Same result as SESSIONPROPERTY
```

Result

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

Summary of actions in order of precedence: The first "Yes" from the left takes precedence.

Table 2-90 Case 4—NUMERIC_ROUNDABORT Option Uses Database Option Unless Explicitly Set

Explicit SET	Set by OLE DB/ ODBC	User Option	Database Default
No	No	Not set	OFF

2.8.3.12 Examples of Displaying Session Properties in Different Clients

Example: This example starts with a new SQL Server instance with all default settings.

No Options Set: Use isql to connect to the new SQL Server instance using (old) DB-Lib

```
C:>    isql   -Usa  -P
1> SELECT @@OPTIONS  As  OptionSettings
2> go
OptionSettings
------ ------------
                          0

1> DBCC USEROPTIONS
2> go
 Set Option          Value
 --------------------------
 textsize          4096
 language          us_english
 dateformat          mdy
```

```
 datefirst        7
(4  row(s) affected)
DBCC execution completed. If DBCC printed error
messages, contact your system administrator.
```

Add ODBC initial settings: The osql utility uses ODBC to connect to SQL Server. This shows the added options set by ODBC (and OLE DB).

```
C:>    osql   -Usa  -P
1> SELECT @@OPTIONS  As  OptionSettings
2> go
OptionSettings
------------------
                5176


1> DBCC USEROPTIONS
2> go
 Set OptionValue
 -------------------------------------
 textsize                   2147483647
 language                   us_english
 dateformatmdy
 datefirst                   7
 ansi_null_dflt_onSET
 ansi_warningsSET
 ansi_paddingSET
 ansi_nullsSET
 concat_null_yields_nullSET
(9  row(s) affected)
DBCC execution completed. If DBCC printed error
messages, contact your system administrator.
```

Other examples of SET, DBCC USEROPTIONS, @@OPTIONS, SESSIONPROPERTY()

SQL
PRINT @@OPTIONS
Result
5496

SQL

```
PRINT @@OPTIONS & 64 -- arithabort bitmask
```

Result

```
64
```

SQL

```
SELECT SESSIONPROPERTY( 'arithabort' )
```

Result

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

Now turn one option off and re-run the display statements.

SQL

```
 SET  arithabort   OFF

DBCC USEROPTIONS
```

Result

```
Set Option                  Value
--------------------------  ------------
textsize                    64512
language                    us_english
dateformat                  mdy
datefirst                   7
quoted_identifier           SET
ansi_null_dflt_on           SET -- arithabort is missing now
ansi_defaults               SET
ansi_warnings               SET
ansi_padding                SET
ansi_nulls                  SET
concat_null_yields_null     SET
(11 row(s) affected)
```

SQL

```
PRINT @@OPTIONS
```

Result

```
5432
```

SQL

```
PRINT @@OPTIONS & 64 -- arithabort bitmask
```

Result

```
0
```

SQL

```
SELECT SESSIONPROPERTY( 'arithabort' )
```

Result

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

SQL

```
SELECT SESSIONPROPERTY( 'CONCAT_NULL_YIELDS_NULL' ) -- Show it's ON
```

Result

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

Here's a nice way to show arithabort setting which uses @@OPTIONS.

SQL

```
PRINT 'ARITHABORT: ' + CASE WHEN @@OPTIONS & 64 > 0 THEN 'ON' ELSE  'OFF' END
```

SQL (cont.)

Result

```
ARITHABORT: OFF
```

SQL

```
SET ARITHABORT ON

PRINT 'ARITHABORT: ' + CASE WHEN @@OPTIONS & 64 > 0 THEN 'ON' ELSE  'OFF' END
```

Result

```
ARITHABORT: ON
```

2.8.4 Default Nullability of New Columns in a Table

This subject seems unduly complex. Leaving everything default as it comes out of the box seems most useful and is certainly easiest, as in the example CREATE TABLE t below. But here are the details for those who enjoy confusing topics.

What I call the *default nullability* setting means that if a user executes CREATE TABLE or ALTER TABLE to add a new column to a table and does not specify either NULL or NOT NULL explicitly, the default nullability setting determines the nullability of the new column, that is, whether it will be created as NULL or NOT NULL.

ANSI SQL-92 standard specifies default nullability to be nullable, that is, default is NULL.

Default nullability is determined by database and session settings. Session setting for ANSI_NULL_DFLT_ON or ANSI_NULL_DFLT_OFF determines the default nullability if either is ON. (Setting one ON sets the other OFF.) Database setting ANSI_NULL_DEFAULT will rule if both session settings are OFF.

Bottom line: ODBC drivers and OLE DB providers set ANSI_NULL_DFLT_ON to ON for each connection, so the Query Analyzer and other clients using these libraries behave with new columns defaulting to nullable.

ANSI_NULL_DFLT_ON will thus be ON unless you explicitly issue either

```
SET ANSI_NULL_DFLT_ON     OFF
  or  SET ANSI_NULL_DFLT_OFF   ON
```

This setting will remain in effect for the rest of your connection unless you change it.

It is suggested that you do not issue either of these statements and so leave the out-of-the-box defaults intact. In this case, use the following CREATE TABLE statement.

```
CREATE TABLE  t (
  col1  INT  NOT NULL,-- col1 will NOT allow NULL and
  col2  INT  NULL    ,-- col2 will allow NULL regardless of settings
  col3  INT          )-- col3 heeds the settings
```

This would result in **col3** being nullable as if it had been created just like col2.

If you do issue either of the two SET statements above, then col3 would be non-nullable as if it had been created like **col1**.

The only way for the ANSI_NULL_DEFAULT database setting to have an effect is if SET ANSI_NULL_DFLT_ON OFF is executed, so this database option seems pretty much useless unless you want to issue that statement, or if you can find a way to connect without using either OLE DB or ODBC.

It should be noted for the record that, according to Books Online, "Microsoft SQL Server 2000 defaults to NOT NULL." So the database option ANSI_NULL_DEFAULT will be found to be OFF, but again, this is overridden by the OLE DB and ODBC drivers turning ON the ANSI_NULL_DFLT_ON option.

2.8.4.1 How to Set and Determine the Current Nullability Settings

The remaining discussion in this section is for completeness and could easily be skipped.

Three levels have a hand in determining the ultimate default nullability of a new column.

Server Configuration This affects session options of logins created after the change.

```
sp_configure  'user options'  ,  1024     — Turns on ANSI_NULL_DFLT_ON
sp_configure  'user options'  ,  2048     — Turns on ANSI_NULL_DFLT_OFF
```

Only one may be ON or both OFF: Setting one ON sets the other OFF.

These seem to have no effect since they assign the SET options of the session, but both ODBC and ODE DB set ANSI_NULL_DFLT_ON to true for each session.

Database Configuration

```
ALTER DATABASE dbname SET  ANSI_NULL_DEFAULT {ON|OFF}
```

Default setting is OFF.

Current database setting is visible with:

```
SELECT  DATABASEPROPERTYEX( 'dbname' ,
'IsAnsiNullDefault'  )
```

Session (Connection) Settings These take precedence if either is ON.

```
SET   ANSI_NULL_DFLT_ON     {ON | OFF}
```

```
SET   ANSI_NULL_DFLT_OFF   {ON | OFF}
```

Only one may be ON or both OFF: Setting one ON sets the other OFF. Also, SET ANSI_DEFAULTS ON includes SET ANSI_NULL_DFLT_ON ON.

Settings of the current session in the current database are visible with:

```
DBCC USEROPTIONS
```

Shows if ANSI_NULL_DFLT_ON or ANSI_NULL_DFLT_OFF is SET. Show effective nullability settings in specified database in current session.

```
SELECT  GETANSINULL ( [ 'dbname' ] )
```

Returns 1 if NULL, 0 if NOT NULL is the effective nullability. This is what is used.

Example:

SQL

```
SELECT  GETANSINULL ( 'pubs' ) --Shows the default
  nullability is NULL in pubs db in this session
```

Result

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

GETANSINULL() result shows what will be used in a CREATE TABLE. Table 2-91 shows how.

Table 2-91 Default Nullability

Session ANSI_NULL_DFLT_ON	Session ANSI_NULL_DFLT_OFF	Default Nullability of New Columns
ON	ON	Impossible (either ON turns other OFF)
ON	OFF	New columns default to **nullable** DATABASE setting is IGNORED
OFF	ON	New columns default to **not nullable** DATABASE setting is IGNORED
OFF	OFF	DATABASE ANSI_NULL_DEFAULT **SETTING RULES**

2.8.5 Collation

A collation determines how sort order, case sensitivity and related issues are handled for columns of string data types, that is **char**, **varchar**, **text**, **nchar**, **nvarchar** and **ntext**.

SQL Server is installed with a default server level collation. SS 2000 default is, "=Dictionary order, case-insensitive, for use with 1252 Character Set.

SQL Server 2K supports different collations for each database down to the level of columns within a table. SQL Server 7.0 allows only a single collation for an instance.

The server level default collation will usually be the collation of every database, and the database default will be the default collation of each table column of string data type.

The COLLATE clause may specify collation for a database or for a column in a table.

A COLLATE clause may be applied at several levels including to a

- database definition,
- column definition in a table or
- string expression

These determine comparison and sorting characteristics. See examples of each below.

New in SQL Server 2K is the capability to create a new database using the COLLATE clause to specify a different collation.

```
CREATE DATABASE  databasename COLLATE  <collation_name>

ALTER DATABASE   databasename COLLATE  <collation_name>
```

See Books Online for restrictions on changing an existing database collation.

Also new with SQL Server 2K is the ability to set a collation for a single column of a table or table variable.

```
CREATE TABLE tablename (
  columnname   columndefinition   COLLATE <collation_name>
... )
```

The code *collation_name* can be a Windows collation name or SQL collation name, and is applicable only for columns of **char, varchar, text, nchar, nvarchar** and **ntext** data types.

For a list of all Windows and SQL collations, execute the following sequence.

```
SQL
SELECT * FROM ::fn_helpcollations()
```

```
Result
```

name	description
Albanian_BIN	Albanian, binary sort
Albanian_CS_AS	Albanian, case-sensitive, accent-sensitive, kanatype-insensitive, width-ins...
...	
Latin1_General_CI_AS	Latin1-General, **case-insensitive**, accent-sensitive, kanatype-insensitive, ...
Latin1_General_CS_AS	Latin1-General, **case-sensitive**, accent-sensitive, kanatype-insensitive, ...
....	

Note: Unicode was designed to eliminate the code page conversion difficulties of the non-Unicode **char**, **varchar** and **text** data types. When you support multiple languages, use the Unicode data types **nchar, nvarchar** and **ntext** for all character data.

Two example collations follow.

Latin1_General_CI_ASCI means case insensitive

Latin1_General_CS_ASCS means case sensitive

Latin1_General is the Latin alphabet used by western European languages. It is also referred to as the **1252 character set**.

Example: Create a **Database** with a specified collation (Case Sensitive).

```
SQL
CREATE DATABASE  mydb COLLATE Latin1_General_CS_AS

USE mydb
go

CREATE TABLE Table1 ( a  INT  ,  A INT )

INSERT INTO Table1   VALUES ( 1 , 2 )

SELECT * FROM Table1  WHERE  a = 1
```

SQL (cont.)

Result

```
a           A
------      ------
1           2
(1 row(s) affected)
```

SQL

```
SELECT * FROM Table1  WHERE  a = 2
```

Result

```
a           A
------      ------

(0 row(s) affected)
```

SQL

```
SELECT * FROM Table1  WHERE  A = 1
```

Result

```
a           A
------      ------

(0 row(s) affected)
```

SQL

```
SELECT * FROM table1
```

Result

```
Server: Msg 208, Level 16, State 1, Line 1
Invalid object name 'table1'.
```

Example: Specify the collation of a string using CASTNotice that CI is for case insensitive and CS is for case sensitive. Without the CAST statement, the database collation is used for comparison.

SQL

```
USE pubs    -- pubs has default case insensitive collation
go
IF 'abc' = 'ABC'   -- We expect TRUE if case insensitive,
  FALSE if case sensitive
           PRINT 'TRUE.  Yes, they compare'
ELSE
           PRINT 'FALSE.  Nope, not the same'
```

Result

TRUE. Yes, they compare

SQL

```
USE mydb    -- mydb was created above with case sensitive
  collation, so it should be FALSE
go
IF 'abc' = 'ABC'   -- We expect TRUE if case insensitive,
  FALSE if case sensitive
           PRINT 'TRUE.  Yes, they compare'
ELSE
           PRINT 'FALSE.  Nope, not the same'
```

Result

FALSE. Nope, not the same

But, the string can be CAST to case insensitive.

SQL

```
IF 'abc' = CAST( 'ABC' as VARCHAR(10) ) COLLATE  Latin1_General_CI_AS
    PRINT 'Yes, they compare'
ELSE
    PRINT 'Nope, not the same'
```

Result

TRUE. Yes, they compare

Example: Create table columns with a specified collation. Overrides database default.

```
CREATE  TABLE t (
     ci VARCHAR(10) COLLATE  Latin1_General_CI_AS
   , cs VARCHAR(10) COLLATE Latin1_General_CS_AS
     )

     INSERT INTO t VALUES ( 'aaa', 'aaa' );
     INSERT INTO t VALUES ( 'AAA', 'AAA' );
Column ci is case insensitive for searches, column cs is case sensitive.
```

SQL

```
SELECT * FROM t WHERE ci = 'aaa'
```

Result

```
ci          cs
----------  ----------
aaa         aaa
AAA         AAA
```

SQL

```
SELECT * FROM t WHERE cs = 'aaa'
```

Result

```
ci          cs
----------  ----------
aaa         aaa
```

Use the string CAST on the column to get case insensitive search.

SQL

```
SELECT * FROM t
WHERE   'aaa'  =  CAST( cs  AS   VARCHAR(10) ) COLLATE  Latin1_General_CI_AS
```

SQL (cont.)

Result

```
ci          cs
----------  ----------
aaa         aaa
AAA         AAA
```