CHAPTER 11

Database Interoperability

Designing databases for interoperability imposes a whole new set of requirements on the database designer over and above those imposed for needs of efficiency, availability, or scalability. This chapter describes a number of issues you should consider when designing a database to be interoperable between multiple database platforms. The idea behind interoperability is to design a database from the start that requires little or no redesigning to work on another database platform. I'll discuss interoperability from the perspective of a user who's designing a SQL Server database that is portable to other database platforms.

Step One: Datatypes

The first step in designing a database with interoperability in mind is to start from the ground up. Thus, whenever you design tables or procedural code that requires datatypes, you should use datatypes that are easily transferable among the major database platforms.

It's important to remember that database platforms can support similarly named datatypes, but their implementation could vary. Please consult Table 11-1 for specific requirements of each platform's datatypes.

Table 11-1. Datatype Compatibility Chart

All Datatypes on All Database Platforms	MySQL v5.1	Oracle 11g	SQL Server 2008	For Interoperability Use
BFILE		X		
BIGINT	X		X	*BIGINT*
BINARY	X		X	*BLOB*
BINARY_FLOAT		X		*FLOAT*
BINARY_DOUBLE		X		*DOUBLE PRECISION*
BIT	X		X	
BIT VARYING, VARBIT				
BLOB	X	X		*BLOB*
BOOL, BOOLEAN	X			*BOOLEAN*
BYTEA				*BLOB*
CHAR, CHARACTER	X	X	X	*CHARACTER*
CHAR FOR BIT DATA				
CLOB		X		*CLOB*
CURSOR			X	

All Datatypes on All Database Platforms	MySQL v5.1	Oracle 11g	SQL Server 2008	For Interoperability Use
DATE	X	X	X	*DATE*
DATETIME	X		X	*TIMESTAMP*
DATETIMEOFFSET			X	*TIMESTAMP*
DATETIME2			X	*TIMESTAMP WITH TIME ZONE*
DEC, DECIMAL	X	X	X	*DECIMAL*
DOUBLE, DOUBLE PRECISION	X	X	X	*FLOAT*
ENUM	X			
FLOAT	X	X	X	*DOUBLE PRECISION*
GEOGRAPHY			X	
GEOMETRY			X	
HIERARCHYID			X	
IMAGE			X	
INT, INTEGER	X	X	X	*INTEGER*
INTERVAL DAY TO SECOND		X		*INTERVAL DAY TO SECOND*
INTERVAL YEAR TO MONTH		X		*INTERVAL YEAR TO MONTH*
LONG		X		
LONGBLOB	X			*BLOB*
LONG RAW		X		*BLOB*
LONGTEXT	X			
MEDIUMBLOB	X			
MEDIUMINT	X			*INT*
MEDIUMTEXT	X			
MONEY			X	
NATIONAL CHARACTER VARYING, NATIONAL CHAR VARYING, NCHAR VARYING,NVARCHAR	X	X	X	*NATIONAL CHARACTER VARYING*
NCHAR, NATIONAL CHAR, NATIONAL CHARACTER	X	X	X	*NATIONAL CHARACTER*
NCLOB		X		*NCLOB*
NTEXT, NATIONAL TEXT			X	*NCLOB*
NVARCHAR2(N)		X		

All Datatypes on All Database Platforms	MySQL v5.1	Oracle 11g	SQL Server 2008	For Interoperability Use
NUMBER	X	X	X	
NUMERIC			X	*NUMERIC*
RAW		X		
REAL	X	X	X	*REAL*
ROWID		X		
ROWVERSION			X	
SERIAL	X			
SET	X			
SMALLDATETIME			X	
SMALLINT	X	X	X	*SMALLINT*
SMALLMONEY			X	
SQL_ VARIANT			X	
TABLE			X	
TEXT	X		X	
TIME	X		X	*TIME*
TIMESTAMP	X	X	X	*TIMESTAMP*
TINYBLOB	X		X	
TINYINT	X		X	
TINYTEXT	X			
UNIQUEIDENTIFIER			X	
UROWID		X		
VARBINARY	X		X	*BLOB*
VARCHAR, CHAR VARYING, CHARACTER VARYING	X	X	X	*CHARACTER VARYING(N)*
VARCHAR2		X		*CHARACTER VARYING*

This table shows you the various datatypes supported by the database platforms discussed in this chapter. Using the chart, you can see the datatypes supported by Microsoft SQL Server. If you need to convert Microsoft SQL Server T-SQL or SQL code to one of the other platforms, use the datatype referenced in the column headed by "For Interoperability Use." Similarly, use the value in this same column when you wish to convert a datatype from one of the other platforms over to SQL Server. Note that when this column is empty, all database platforms support the datatype on the row of the table.

Step Two: Identifier Rules

When you want to create new objects within a SQL Server database (or any database for that matter), you must abide by the identifier rules for that platform. Identifiers are commonly used to name the columns of a table or variables in procedural code, such as a user-defined function or a stored procedure. It’s important to learn the ins and outs of identifiers. Table 11-2 contrasts the identifier rules for SQL Server 2008 as compared to several other popular database platforms.

Table 11-2. Platform-Specific Rules for Regular Object Identifiers (Excludes Quoted Identifiers)

Characteristic	Platform	Specification
Identifier size	MySQL	64 characters; aliases may be 255 characters
	Oracle	30 bytes (number of characters depends on the character set); database names are limited to 8 bytes; database links are limited to 128 bytes
	SQL Server	128 characters (temp tables are limited to 116 characters)
Identifier may contain	MySQL	Any number, character, or symbol. Cannot be composed entirely of numbers.
	Oracle	Any number, character, and the underscore (_), pound (#), and dollar ($) symbols though the last two are discouraged. Database links may also contain a period (.) symbol.
	SQL Server	Any number, character, and the underscore (_), at sign (@), pound (#), and dollar ($) symbols
Identifier must begin with	MySQL	A letter or number. Cannot be composed entirely of numbers.
	Oracle	A letter
	SQL Server	A letter, underscore (_), at sign (@), or pound (#)
Identifier cannot contain	MySQL	Period (.), slash (/), or ASCII(0) and ASCII(255). Quote (‘) and double-quote (“) are only allowed in quoted identifiers. Should not end with space characters.
	Oracle	Spaces, double-quotes (“), or special characters
	SQL Server	Spaces or special characters
Allows quoted identifiers	MySQL	Yes
	Oracle	Yes
	SQL Server	Yes
Quoted identifier symbol	MySQL	Quote (‘) or double-quote (“) in ANSI compatibility mode
	Oracle	Double-quote (“)
	SQL Server	Double-quote (“) or brackets ([ ]); brackets are preferred
Identifier may be reserved	MySQL	Quoted identifier only
	Oracle	Quoted identifier only
	SQL Server	Quoted identifier only
Schema addressing	MySQL	Database.object
	Oracle	Schema.object
	SQL Server	Server.database.schema.object

Identifier must be unique	MySQL	Yes
	Oracle	Yes
	SQL Server	Yes
Case Sensitivity	MySQL	Only if underlying file system is case sensitive, e.g. Mac OS or Unix. Triggers, logfile groups, and tablespaces are always case sensitive.
	Oracle	No, by default but can be changed
	SQL Server	No, by default but can be changed
Other rules	MySQL	May not contain numbers only
	Oracle	Database links are limited to 128 bytes and may not be quoted identifiers
	SQL Server	Microsoft is commonly using brackets rather than double-quotes for quoted identifiers.

You can break, or at least bend, many of the rules described in Table 11-2 through the use of quoted identifiers. Quoted identifiers are object names encapsulated within a special delimiter, usually double quotes. SQL Server also commonly use brackets ([ ]) as delimiters. You can use quoted identifiers to put normally illegal characters into an object name or to bestow an object name that's normally illegal, such as a reserved word. All the platforms discussed in this chapter support quoted identifiers. Once you have declared an object as a quoted identifier, you should always reference it using its quoted identifier name.

Tip Follow these three rules of thumb when naming objects and choosing identifiers. First, avoid any kind of reserved word in identifiers. Second, avoid specific national characters in identifiers. Finally, avoid key words in identifiers; that is, words that might become reserved words in a future release of the product.

Step Three: Basic SQL Statements

In this section, I'll describe the best way to create interoperable versions of the standard SQL statements: DELETE, INSERT, SELECT, and UPDATE. Note that a full discussion of the syntax and usage of each command is beyond the scope of this chapter. Instead, I focus on the interoperability challenges between SQL Server and other database platforms.

Note Italic syntax shown in the following tables isn't interoperable between database platforms.

Thus, I'll describe the most interoperable syntax to create each database object, and follow that up with important notes and details about variations on each database platform. The assumption is that you might want to move code both *from* SQL Server and *to* SQL Server, so I explain both scenarios.

To learn the full syntax and usage of any of these statements, please refer to SQL Server Books Online.

The DELETE Statement

The DELETE statement erases records from a specified table or tables. The only 100 percent safe and interoperable form of the DELETE statement follows this syntax:

```
DELETE FROM table_name
[WHERE search_condition]
```

Using this syntax, the FROM table_name clause identifies the table where one or more rows will be deleted. The table_name assumes the current schema if one isn't specified. You can alternately specify a view name, as long as the view is built on a single table. The WHERE search_condition clause defines the search criteria for the DELETE statement, using one or more search conditions to ensure that only the target rows are deleted. Any legal WHERE clause is acceptable in the statement. Typically, these criteria are evaluated against each row of the table before the deletion occurs.

Table 11-3 compares the DELETE statement syntax on several major platforms. Note that the syntax displayed in italics is platform specific and cannot interoperate on other database platforms.

Table 11-3. Comparison of DELETE Statement Syntax

Database Platform	Syntax
SQL Server	***[WITH cte_expression [, …]]*** DELETE [TOP (number) [PERCENT]] [FROM] table_name [[AS] alias] ***[WITH (hint […])]*** ***[OUTPUT expression INTO {@table_variable | output_table} [(column_list [,…])]]*** ***[FROM table_source [,...]]*** [[{INNER | CROSS | [{LEFT | RIGHT | FULL] OUTER}] JOIN joined_table ON condition][,...] [WHERE search_condition | WHERE CURRENT OF [GLOBAL] cursor_name] ***[OPTION (hint [,...n])]***
MySQL	DELETE [LOW_PRIORITY] [QUICK] [table_name[.*] [,...]] {***FROM table_name[.*] [,...] | [USING table_name[.*] [,...]***] } [WHERE search_condition] ***[ORDER BY clause]*** ***[LIMIT nbr_of_rows]***
Oracle	DELETE [FROM] {table_name | ***ONLY (table_name)***} [alias] [{***PARTITION (partition_name) |*** ***SUBPARTITION (subpartition_name)}] |*** ***(subquery [WITH {READ ONLY |*** ***CHECK OPTION [CONSTRAINT constraint_name]}]) |***

	TABLE (collection_expression) [(+)] } *[hint]* [WHERE *search_condition*] *[RETURNING expression [,...] INTO variable [,...]]* *[LOG ERRORS [INTO [schema.]table_name]* *[(simple_expression)]* *[REJECT LIMIT {UNLIMITED |int }]]*

SQL Server

Most significantly, SQL Server also allows a second FROM clause to allow JOIN constructs, as shown in the syntax in Table 11-3. The following clauses aren't transportable to other database platforms:

```
TOP number [PERCENT]
FROM table_source [,...]  (second FROM clause)
[{INNER | CROSS | [{LEFT | RIGHT | FULL] OUTER}]
JOIN joined_table ON condition [,...]
WHERE CURRENT OF [GLOBAL] cursor_name
OPTION (query_hint[,...n])
```

For example, the following interoperable DELETE statement uses a rather complex subquery to erase all the sales records of computer books:

```
DELETE sales
WHERE title_id IN
  (SELECT title_id
  FROM titles
  WHERE type = 'computer')
```

However, SQL Server allows a more elegant construction using a FROM clause and a JOIN clause, which isn't transportable to other database platforms:

```
DELETE
FROM sales AS
INNER JOIN titles AS t ON s.title_id = t.title_id
  AND type = 'computer'
```

The second DELETE clause would be illegal on any other database platform.

Also note that Common Table Expressions (CTE's), while now common on many database platforms and very useful, are not part of the standard. So avoid CTEs if you're trying to write interoperable code.

MySQL

MySQL allows a number of extensions to the basic DELETE statement, but doesn't support several clauses allowed on SQL Server, such as the WHERE CURRENT OF clause. The following clauses are legal on MySQL, but illegal on SQL Server:

```
USING table_name[.*] [,...]
[ORDER BY clause]
```

```
[LIMIT nbr_of_rows]
```

As you can see in the syntax, MySQL also supports a FROM clause for building more effective JOIN constructs. The USING table_name[.*] [,...] clause substitutes the table or tables before the FROM clause and those after the FROM clause.

The ORDER BY clause specifies the order in which rows will be deleted. This is useful only in conjunction with LIMIT. The LIMIT nbr_of_row clause places an arbitrary cap on the number of records deleted before control is passed back to the client. You can simulate this behavior on SQL Server by using a DELETE statement with a SELECT TOP statement as a WHERE clause condition.

MySQL allows deletion from more than one table at a time. For example, the following two DELETE statements are functionally equivalent:

```
DELETE sales FROM clients, sales
WHERE clients.clientid = sales.clientid
  AND sales.salesdate BETWEEN '20040101' AND '20071231'

DELETE FROM sales USING clients, sales
WHERE clients.clientid = sales.clientid
  AND sales.salesdate BETWEEN '20040101' AND '20071231'
```

MySQL does a lot of things behind the scenes to speed up processing. For example, when MySQL is in AUTOCOMMIT mode, it even substitutes a TRUNCATE statement for a DELETE statement without a WHERE clause, because TRUNCATE is faster. The speed of a MySQL delete operation is directly related to the number of indexes on the table and the available index cache. You can speed up delete operations by executing the command against tables with few or no indexes, or by increasing the size of the index cache.

Oracle

When moving between SQL Server and Oracle, it's best to stick with the basic DELETE statement. Oracle allows lots of unique behavior in its implementation of the DELETE statement, such as deleting from materialized views, nested subqueries, and partitioned views and tables, as follows:

```
ONLY (table_name)
PARTITION (partition_name)
SUBPARTITION (subpartition_name)
Subquery [WITH {READ ONLY | CHECK OPTION [CONSTRAINT constraint_name]}]
TABLE (collection_expression) [ (+) ]
LOG ERRORS …

RETURNING expression [,...] INTO variable [,...]
```

Oracle has a lot of unique clauses, such as the clause (subquery [WITH {READ ONLY | CHECK OPTION [CONSTRAINT constraint_name] }]), which specifies that the target for deletion is a nested subquery, not a table, view, or other database object. In addition to a standard subquery (without an ORDER BY clause), the parameters of this clause are the WITH READ ONLY subclause, which specifies that the subquery cannot be updated. In addition, the WITH CHECK OPTION subclause tells Oracle to abort any changes to the deleted table that wouldn't appear in the result set of the subquery. The CONSTRAINT constraint_name subclause tells Oracle to restrict changes further based upon a specific constraint identified by constraint_name. You can simulate this behavior on SQL Server using T-SQL procedural code.

You can apply the delete operation to a table-like expression using the TABLE (collection_expression) [(+)] clause, and to a specific partition or subpartition using PARTITION partition_name and SUBPARTITION subpartition_name, respectively. You can query and manipulate specific partitions by using the $PARTITION range function. $PARTITION returns an integer that designates the partition number that you want to query or manipulate. For example, use the following code to select all the rows that are in partition 2 of the Sales_History table, which uses the Xact_range_Fcn partition function based upon the Sales_Date column:

```
SELECT * FROM Production.Sales_History
WHERE $PARTITION.Xact_Range_Fcn(Sales_Date) = 2 ;
```

The RETURNING expression retrieves the rows affected by the command, where DELETE normally only shows the number of rows deleted. The INTO variable clause specifies the variables into which the values are stored that are returned as a result of the RETURNING clause. There must be a corresponding variable for every expression in the RETURNING clause. The *LOG ERRORS* clause tells Oracle to store DML error information plus the column values of affected rows and, optionally, if *REJECT LIMIT* is met or exceeded then rolling back the entire transaction.

The INSERT Statement

The INSERT statement inserts one or more rows into a table or view. The basic and most interoperable form of the statement follows this syntax:

```
INSERT INTO {table_name | view_name} [(column1 [,...] )]
{VALUES (value1 [,...]) | SELECT statement }
```

The clause INSERT INTO table_name (col1, col2, col3) specifies the name of the table (in this case, table_name) and the specific columns within the table that will receive the inserted values (in this case, col1, col2, and col3). You may alternately choose the VALUES (value1, value2, value3) clause, or issue a SELECT statement with the same number and datatype of columns as specified in the column list.

Table 11-4 compares the INSERT statement syntax on several major platforms. Note that the syntax displayed in italics is platform specific and cannot interoperate on other database platforms.

Table 11-4. Comparison of INSERT Statement Syntax

Database Platform	Syntax
SQL Server	***[WITH cte_expression [, …]]*** INSERT ***[TOP (number) [PERCENT]]*** [INTO] `table_name` [(`column1` [,...])] [***OUTPUT expression INTO {@table_variable | output_table} [(column_list [,…])]***] {[DEFAULT] VALUES | VALUES (`variable2` [,...]) | SELECT_`statement` | ***EXEC[UTE] proc_name { [[@param =] value {[OUTPUT]} [,...]]*** }

MySQL	INSERT *[LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]*
	[INTO] [[database_name.]owner.] table_name [(column1 [,...])]
	{VALUES ({value1 | DEFAULT} [,...]) | SELECT_statement |
	SET [*ON DUPLICATE KEY UPDATE*] column1=value1, column2=value2 [,...]}

Oracle	-- standard INSERT statement INSERT [INTO] {table_name [*[SUB]PARTITION { (prtn_name)*| (key_value) }] | (subquery) *[WITH {READ ONLY | CHECK OPTION [CONSTRAINT constr_name] }]* | TABLE (collection) [(+)] } [alias] [(column1 [,...])] {VALUES (value1[,...]) *[RETURNING expression1 [,...] INTO variable1 [,...]]* | SELECT_statement *[WITH {READ ONLY | CHECK OPTION [CONSTRAINT constr_name]}* } *[LOG ERRORS [INTO [schema.]table_name] [(expression)] [REJECT LIMIT { int | UNLIMITED}]]* -- conditional INSERT statement *INSERT {[ALL | FIRST]} WHEN condition* *THEN standard_insert_statement* *ELSE standard_insert_statement* *[LOG ERRORS [INTO [schema.]table_name] [(expression)] [REJECT LIMIT { int | UNLIMITED}]]*

SQL Server

SQL Server supports one unique extension to the basic INSERT statement that isn’t interoperable: the ability to insert the results from stored procedures and extended procedures directly into the target table. The following clauses aren’t transportable to other database platforms:

WITH cte_expression

TOP (number) [PERCENT]

OUTPUT expression INTO…

EXEC[UTE] proc_name { [[@param =] value {[OUTPUT]} [,...]] }

The clause EXEC[UTE] proc_name { [[@param =] value {[OUTPUT]} [,...]] } tells SQL Server to execute a dynamic T-SQL statement or a routine—such as stored procedure, a remote procedure call (RPC), or an extended stored procedure—and insert the result set into a local table. proc_name is the name of the stored procedure you wish to execute, and @param is the parameter(s) of the routine (the @ sign is required). You can also assign a value to the parameters, and optionally designate the parameter as an OUTPUT parameter. The columns returned by the result set must match the datatype of the columns in the target table.

MySQL

MySQL supports several INSERT syntax options that aren’t interoperable:

```
LOW_PRIORITY | DELAYED | HIGH_PRIORITY
IGNORE
SET column1 = value1 [,...]
```

Most of MySQL’s specialized features engender MySQL’s reputation for quick response times to the end user using the LOW_PRIORITY and DELAYED clauses. LOW_PRIORITY defers the execution of the insert operation until no other clients are reading from the table, possibly resulting in a long wait. (You shouldn’t use this option on MyISAM tables.) The DELAYED clause enables the client to continue working, even when the insert operation hasn’t completed on MySQL. *HIGH_PRIORITY* overrides the LOW_PRIORITY setting but doesn’t otherwise effect behavior.

The IGNORE operation tells MySQL not to insert duplicate values. If MySQL encounters duplicate records and this clause isn’t used, the INSERT will fail. The SET column = value clause is an alternative syntax that allows you to specify values for target columns by name.

MySQL behaves quirkily when it comes to datatype mismatches and oversized inserts on CHAR, VARCHAR, TEXT, BLOB, and numeric columns. MySQL trims any portion of a value that has a size or datatype mismatch. Thus, inserting '19.66 Y' into a decimal datatype column inserts '19.66' and trims '...Y' from the value. If you attempt to insert a numeric value into a column that’s beyond the range of the column, MySQL will trim the value. Inserting an illegal time or date value in a column results in a zero value for the target column. Inserting the ten-character string into a CHAR(5) column inserts just the first five characters of the string into the column.

Oracle

Oracle’s implementation of the INSERT statement allows inserts into many tables simultaneously, conditional inserts, and data insertion into a given table, view, partition, subpartition, or object table:

```
PARTITION (partition_name)
SUBPARTITION (subpartition_name)
Subquery [WITH {READ ONLY | CHECK OPTION [CONSTRAINT constraint_name]}]
TABLE (collection_expression) [ (+) ]
LOG ERRORS…
RETURNING expression [,...] INTO variable [,...]
INSERT {[ALL | FIRST]} WHEN...
```

Oracle has a lot of unique clauses for this statement, such as the PARTITION (partition_name) clause and SUBPARTITION (subpartition_name) clause, which identify a specific partition or subpartition of a table where the insert operation should occur.

The (Subquery [WITH {READ ONLY | CHECK OPTION [CONSTRAINT constraint_name]}]) clause specifies that the target for record insertion is a nested subquery, not a table, view, or other database object. Subqueries in the VALUES clause are interoperable with SQL Server, but not with the WITH subclause.

In addition to a standard subquery (without an ORDER BY clause), the parameters of this clause are the WITH READ ONLY subclause, which specifies that the subquery cannot be updated. In addition, the WITH CHECK OPTION subclause tells Oracle to abort any changes to the table concerned that wouldn't appear in the result set of the subquery. The CONSTRAINT constraint_name subclause tells Oracle to restrict changes further based upon a specific constraint identified by constraint_name.

You can apply the insert operation to a table-like expression using the TABLE (collection_expression) [(+)] clause. You should remove this clause when you move the code to SQL Server, because there's no corollary on SQL Server.

The RETURNING expression retrieves the rows affected by the command, where INSERT normally only shows the number of rows deleted. The INTO variable clause specifies the variables into which the values returned as a result of the RETURNING clause are stored. There must be a corresponding variable for every expression in the RETURNING clause.

The conditional and/or multitable INSERT is another of Oracle's unique capabilities. Oracle can use the optional ALL keyword to perform an insert operation on many tables simultaneously. Although this command is beyond the scope of this book, the syntax of the INSERT statement is included so that you can recognize it should you ever see it and need to translate it to SQL Server. You can simulate this behavior on SQL Server only via T-SQL procedural code.

The *LOG ERRORS* clause tells Oracle to store DML error information plus the column values of affected rows and, optionally, if *REJECT LIMIT* is met or exceeded then rolling back the entire transaction.

The SELECT Statement

The SELECT statement retrieves rows, columns, and derived values from one or many tables of a database. The basic and most interoperable form of the statement follows this syntax:

```
SELECT [{ALL | DISTINCT}] select_item [[AS] alias] [,...]
FROM {table_name  [[AS] alias] | view_name [[AS] alias]} [,...]
[ [ {INNER | FULL | LEFT [OUTER] | RIGHT [OUTER]} ] JOIN join_condition ]
[WHERE [NOT] search_condition] [ {AND | OR } [ NOT] search_condition [...] ]
[GROUP BY group_by_columns  [HAVING search_condition] ]
[ORDER BY {order_expression [ASC | DESC]} [,...] ]
```

Table 11-5 compares the SELECT statement syntax on several major platforms. Note that the syntax displayed in italics is platform specific and cannot interoperate on other database platforms.

Table 11-5. Comparison of SELECT Statement Syntax

Database Platform	**Syntax**
SQL Server	***[WITH cte_expression [,…]]*** SELECT {[ALL | DISTINCT] | ***[TOP number [PERCENT][WITH TIES]]***} select_item [AS alias] ***[INTO new_table_name]*** [FROM {[rowset_function | table1 [,...]} [AS alias]] [[join type] JOIN table2 {[ON join_condition]]

	[WHERE search_condition] [GROUP BY [GROUPING SETS] {grouping_column [,...]| ALL}] [WITH { CUBE | ROLLUP }] [HAVING search_condition] [ORDER BY order_by_expression [ASC | DESC]] [COMPUTE {aggregation (expression)} [,...] [BY expression [,...]]] [FOR {BROWSE | XML { RAW | AUTO | EXPLICIT} [, XMLDATA][, ELEMENTS][, BINARY base64]] [OPTION (<hint> [,...])]
MySQL	SELECT [DISTINCT | DISTINCTROW | ALL] [STRAIGHT_JOIN][{SQL_SMALL_RESULT | SQL_BIG_RESULT}][SQL_BUFFER_RESULT] [{SQL_CACHE | SQL_NO_CACHE}] [SQL_CALC_FOUND_ROWS] [HIGH_PRIORITY] select_item AS alias [,...] [INTO {OUTFILE | DUMPFILE | variable [,…]} 'file_name' options] [FROM table_name AS alias [,...] [{ USE INDEX (index1 [,...]) | IGNORE INDEX (index1 [,...]) }] [join type][JOIN table2] [ON join_condition] [WHERE search_condition] [GROUP BY {unsigned_integer | column_name | formula} [ASC | DESC] [,...] [WITH ROLLUP]] [HAVING search_condition] [ORDER BY {unsigned_integer | column_name | formula} [ASC | DESC] [,...]] [LIMIT { [offset_position,] number_of_rows] | number_of_rows OFFSET offset_position}] [PROCEDURE procedure_name (param [,...])] [{FOR UPDATE | LOCK IN SHARE MODE}] ;

Oracle	[WITH cte_name AS (subquery) [,...]]
	SELECT { {[ALL | DISTINCT]} | [UNIQUE]} [optimizer_hints]
	  select_item [AS alias] [,...]
	[INTO {variable [,...] | record}]
	FROM {[ONLY] {[schema.][table_name| view_name |
	  materialized_view_name]}[@database_link][AS [OF] {SCN | TIMESTAMP} expression] |
	  subquery [WITH {READ ONLY | CHECK OPTION [CONSTRAINT constraint_name]}] |
	  [[VERSIONS BETWEEN {SCN | TIMESTAMP} {exp | MINVALUE AND
	    {exp | MAXVALUE}] AS OF {SCN | TIMESTAMP} expression] |
	  TABLE (nested_table_column) [(+)]
	   {[PARTITION (partition_name) | SUBPARTITION (subpartition_name)]}
	  [SAMPLE [BLOCK] [sample_percentage] [SEED (seed_value)]} [AS alias]
	   [,...]
	[[join_type] JOIN join_condition [PARTITION BY expression [,...]]]

	[WHERE *search_condition*] [{AND | OR} *search_condition* [,...]] ***[[START WITH value] CONNECT BY [PRIOR] condition]]*** [GROUP BY *group_by_expression* [HAVING *search_condition*]] ***[MODEL model_clause]*** [ORDER [*SIBLINGS*] BY *order_expression* {[ASC | DESC]} ***{[NULLS FIRST | NULLS LAST]}]*** ***[FOR UPDATE [OF [schema.][table.]column] [,...]*** ***{[NOWAIT | WAIT (integer)]}]***

SQL Server

SQL Server supports most of the basic elements of the ANSI SELECT statement, including all the various join types. SQL Server offers several variations on the SELECT statement, including optimizer hints, the INTO clause, the TOP clause, GROUP BY variations, COMPUTE, and WITH OPTIONS. (Note that Oracle supports both the GROUPING SETS clause and Common Table Expressions, but many other database platforms do not support these clauses.)

When writing interoperable SELECT statements on SQL Server, avoid the following clauses:

```
WITH cte_expression

TOP number [PERCENT] [WITH TIES]
INTO new_table_name ]
GROUP BY GROUPING SETS
WITH { CUBE | ROLLUP }
COMPUTE {aggregation (expression)} [,...]    [BY expression [,...] ]
FOR {BROWSE | XML { RAW | AUTO | EXPLICIT}[, XMLDATA][, ELEMENTS]
  [, BINARY base64]
OPTION ( <hint> [,...])
```

When moving SELECT statements from SQL Server with these clauses to other database platforms, you'll often have to build elaborate statements or even programs in the native procedural language to implement a workaround. There are no silver bullets for substituting specific interoperable code for each of the different clauses that are unique to SQL Server.

The one exception to this rule is that most database platforms support hints of a similar nature, though the command itself is likely to be different.

MySQL

MySQL's implementation of SELECT includes a few unique clauses, such as the INTO clause, the LIMIT clause, and the PROCEDURE clause, among others. The following MySQL clauses aren't interoperable with SQL Server:

```
[STRAIGHT_JOIN][ {SQL_SMALL_RESULT | SQL_BIG_RESULT} ][SQL_BUFFER_RESULT]
```

```
  [ {SQL_CACHE | SQL_NO_CACHE} ] [SQL_CALC_FOUND_ROWS]
  [HIGH_PRIORITY]
INTO {OUTFILE | DUMPFILE} 'file_name' options
{ USE INDEX (index1 [,...]) | IGNORE INDEX (index1 [,...]) }
  [LIMIT [[offset_position,] number_of_rows] ]
  [PROCEDURE procedure_name (param [,...] ) ]
{FOR UPDATE | LOCK IN SHARE MODE}
```

You can couple many performance options with a MySQL SELECT statement, similar to hints on SQL Server. The STRAIGHT_JOIN clause forces the optimizer to join tables in the exact order they appear in the FROM clause. The SQL_SMALL_RESULT clause and SQL_BIG_RESULT clause tell the query optimizer to expect a small or large result set, respectively, on a query containing a GROUP BY clause or a DISTINCT clause. As a result of the hint, MySQL builds a temporary work table either in memory (quickly, using SQL_SMALL_RESULT) or on disk (less quickly, using SQL_BIG_RESULT). The SQL_BUFFER_RESULT clause forces the result set into a temporary table so that MySQL can free table locks earlier and speed the result set to the client. Finally, the HIGH_PRIORITY gives the query a higher priority than other statements that modify data within the table. You should use this only for special, high-speed queries.

The FROM ... USE INDEX | IGNORE INDEX clause tells MySQL that the query should use only the named index, or without using one or more indexes, respectively.

The INTO {OUTFILE | DUMPFILE} 'file_name' clause writes the result set of the query to a host file named file_name using the OUTPUT option. The DUMPFILE variation writes a single continuous stream of data to a specified host file. The dumpfile contains no column terminations, line terminations, or escape characters, and is best suited for Binary Large Objects (BLOBs). In addition, there are a couple optional parameters with the INTO clause. The LIMIT subclause constrains the number of rows returned by the query, starting at the record numbered offset_position and returning number_of_rows. You have to provide at least one integer value, which is the number of records to return, and a starting record of zero is assumed. Alternately, you can use the PROCEDURE procedure_name (param [,...]) names with an external C or C++ procedure that processes the data in the result set.

The final optional clause allows the choice of FOR UPDATE or LOCK IN SHARE MODE. The FOR UPDATE clause specifies a write lock on the rows returned by the query on an InnoDB table or Berkeley Database (BDB) table. The LOCK IN SHARE clause specifies read locks on the rows returned by the query so that other users may see the rows, but may not modify them.

MySQL supports the following additional types of JOIN syntax: CROSS JOIN, STRAIGHT_JOIN, and NATURAL JOIN.

MySQL implements something like a SELECT statement called the HANDLER statement. The HANDLER statement is a high-speed SELECT statement, and should be converted to a SELECT statement when moving queries to SQL Server from MySQL.

Oracle

Oracle allows a large number of extensions to the SELECT statement. The following code lists clauses that are specific to Oracle:

```
WITH query_name AS (subquery) [,...]
```

```
INTO {variable [,...] | record}
FROM {[ONLY] ... [@database_link] [AS [OF] {SCN | TIMESTAMP} expression] |
  subquery [WITH {READ ONLY | CHECK OPTION [CONSTRAINT constraint_name]}] |
    [[VERSIONS BETWEEN {SCN | TIMESTAMP}
      {exp | MINVALUE AND {exp | MAXVALUE}] AS OF {SCN | TIMESTAMP} expression] |
  TABLE (nested_table_column) [(+)]
    {[PARTITION (partition_name) | SUBPARTITION (subpartition_name)]}
    [SAMPLE [BLOCK] [sample_percentage] [SEED (seed_value)]}
JOIN ... [PARTITION BY expression [,...] ]
[START WITH value] CONNECT BY [PRIOR] condition
MODEL clause
ORDER [SIBLINGS] BY order_expression ... {[NULLS FIRST | NULLS LAST]}
FOR UPDATE [OF [schema.][table.]column] [,...] {[NOWAIT | WAIT (integer)]}
```

The WITH query_name AS (subquery) references a common table expression, which is essentially a named subquery that can be referenced frequently. The INTO clause retrieves a result set into a set of PL/SQL variables or a PL/SQL record.

The FROM clause in Oracle has a variety of options. The ONLY keyword applies to a hierarchical view where you want to retrieve records only from the named view. (Oracle supports hierarchies of tables and views where each higher-level object in the hierarchy contains all the records of objects lower in the hierarchy.) The @database_link clause references a remote Oracle database instance. The AS [OF] {SCN | TIMESTAMP} expression clause implements a flashback query that applies system change numbers (SCNs) to each record that existed at the time of the SCN or TIMESTAMP. Because SQL Server doesn't support flashback queries, you'll have to take an alternative strategy to reproduce this capability on SQL Server. The same applies to the VERSIONS BETWEEN subclause. The VERSIONS BETWEEN subclause defines a flashback query clause to retrieve the changes made to data from a table, view, or materialized view over time.

The TABLE clause is necessary when querying a hierarchically declared nested table. The PARTITION and SUBPARTITION restricts a query to the specified partition or a partition of the table. In essence, rows are retrieved only from the named partition, not the entire table. The SAMPLE clause tells Oracle to select records at random using a random sampling of rows, either as a percentage of rows or blocks, within the result set instead of the whole table. The JOIN clause allows a PARTITION subclause.

The WHERE ... [[START WITH value] CONNECT BY [PRIOR] condition] clause allows control of hierarchical queries across multiple tables in the hierarchy. When translating hierarchical queries from Oracle to SQL Server, you'll most likely have to account for this by doing the following:

* Using a different database design on the SQL Server database
* Using T-SQL procedural code
* Using Common Table Expressions (CTE) on SQL Server

Oracle has a couple variations of the ORDER BY clause that aren't interoperable with SQL Server. The SIBLINGS subclause is used for hierarchical queries. The NULL FIRST and NULL LAST subclauses specify that the records containing NULLs should appear either first or last, respectively.

The FOR UPDATE clause exclusively locks the records of the result set so that other users cannot lock or update them until the transaction is completed. The NOWAIT and

WAIT keywords return control immediately if a lock already exists, or wait integer seconds before returning control, respectively.

Oracle Database 10*g* features a powerful new feature, called MODEL, which enables spreadsheet-like result sets from a SELECT statement. The MODEL clause is a syntactically complex clause. You can reproduce many of its features using SQL Server 2008's PIVOT and UNPIVOT clauses. MODEL, unlike PIVOT, allows you to insert or update values back into the base table.

Note It's well known that cursors on SQL Server can have a profound impact on performance. However, this caveat doesn't apply to Oracle. The reason that cursors don't slow down Oracle's performance is that all SELECT statements execute an implicit cursor behind the scenes. Thus, Oracle SELECT statements and cursors are virtually the same thing. SQL Server, on the other hand, uses different engine operations to execute queries and cursors, thus resulting in a possible performance hit with cursors.

The UPDATE Statement

The UPDATE statement alters values within existing records of a table. The most interoperable syntax for the UPDATE statement follows:

```
UPDATE {table_name | view_name}
SET column_name = { DEFAULT | NULL | scalar_expression} [,...] }
WHERE {search_condition | CURRENT OF cursor_name}
```

Table 11-6 compares the UPDATE statement syntax on several major platforms. Note that the syntax displayed in italics is platform specific and cannot interoperate on other database platforms.

Table 11-6. Comparison of UPDATE Statement Syntax

Database Platform	**Syntax**
SQL Server	UPDATE {*table_name* | *view_name* | *rowset*} ***[WITH (hint1, hint2 [,...])]*** SET {*column_name* = {DEFAULT | NULL | *scalar_expression* } | *variable_name* = *scalar_expression* | *variable_name* = *column_name* = *scalar_expression* } [,...] ***[FROM {table1 | view1 | nested_table1 | rowset1} [,...]]*** [AS *alias*] [JOIN {*table2* [,...]}] WHERE {*conditions* | CURRENT OF [GLOBAL] *cursor_name*} ***[OPTION (hint1, hint2 [,...])]***

MySQL	UPDATE ***[LOW PRIORITY] [IGNORE]*** *table_name*
	SET *column_name* = {*scalar_expression*}
	[,...]

	WHERE *search_conditions* [ORDER BY *column_name1* [{ASC | DESC}] [,...]] ***[LIMIT integer]***
Oracle	UPDATE [***ONLY***] { [*schema*.]{*view_name* | *materialized_view_name* | *table_name*} ***[@database_link] [alias]*** ***{[PARTITION (partition_name)] |*** ***[SUBPARTITION (subpartition_name)]}*** ***| subquery [WITH {[READ ONLY] | [CHECK OPTION [CONSTRAINT constraint_name]]*** ***| [TABLE (collection_expression_name) [(+)]]*** } SET {*column_name1* [,...] = {*expression* [,...] | *subquery*} | VALUE [(*alias*)] = { *value* | (*subquery*)}, {*column_name1* [,...] = {*expression* [,...] | *subquery*} | VALUE [(*alias*)] = { *value* | (*subquery*)}, [,...] WHERE *search_conditions* | CURRENT OF *cursor_name*} ***RETURNING expression [,...] INTO variable [,...];***

SQL Server

SQL Server supports a couple unique clauses within the UPDATE statement, such as the WITH clause, query hints using the OPTION clause, and a secondary FROM clause:

```
WITH (hint1, hint2 [,...])
FROM {table1 | view1 | nested_table1 | rowset1} [,...] [JOIN {table2 [,...]}]
OPTION (hint1, hint2 [,...])
```

To achieve interoperability with other database platforms, you should remove these clauses from your UPDATE statement.

MySQL

MySQL allows a number of extensions over the basic UPDATE statement, but doesn’t support several clauses allowed on SQL Server, such as the WHERE CURRENT OF clause. The following clauses are legal on MySQL, but illegal on SQL Server:

```
LOW_PRIORITY
IGNORE
ORDER BY clause [LIMIT nbr_of_rows]
```

MySQL allows you to declare transaction priority compared to other transactions using the LOW_PRIORITY and IGNORE clauses. The LOW_PRIORITY clause tells MySQL to perform the update operation only when no other users have a lock on the table. The IGNORE clause tells MySQL to ignore any duplicate key errors generated by constraint violations. You can simulate this behavior using SQL Server hints.

The ORDER BY clause specifies the order that rows will be updated. This is useful only in conjunction with LIMIT. The LIMIT nbr_of_rows clause places an arbitrary cap on the number of records updated before control is passed back to the client. You can simulate this behavior on SQL Server by using an UPDATE statement with a SELECT TOP statement as a WHERE clause condition.

The following example limits the UPDATE to the first ten records encountered in the inventory table according to inventory_id. Also, the values of wholesale_price and retail_price are updated:

```
UPDATE inventory
SET wholesale_price = price * .85,
   retail_price = price + 1
ORDER BY inventory_id
LIMIT 10;
```

Oracle

When moving between Oracle and SQL Server, it's best to stick with the basic UPDATE statement. Oracle allows lots of unique behavior in its implementation of the UPDATE statement, such as deleting from materialized views, nested subqueries, and partitioned views and tables, as follows:

```
ONLY (table_name)
PARTITION (partition_name)
SUBPARTITION (subpartition_name)
Subquery [WITH {READ ONLY | CHECK OPTION [CONSTRAINT constraint_name]}]
TABLE (collection_expression) [ (+) ]
RETURNING expression [,...] INTO variable [,...]
```

Oracle has a lot of unique clauses, such as the clause (Subquery [WITH {READ ONLY | CHECK OPTION [CONSTRAINT constraint_name] }]), which specifies that the record to be updated is a result set of a nested subquery, not a table, view, or other database object. In addition to a standard subquery (without an ORDER BY clause), the parameters of this clause are the WITH READ ONLY subclause, which specifies that the subquery cannot be updated. In addition, the WITH CHECK OPTION subclause tells Oracle to abort any changes to the updated table that wouldn't appear in the result set of the subquery. The CONSTRAINT constraint_name subclause tells Oracle to restrict changes further, based upon a specific constraint identified by constraint_name. You can simulate this behavior on SQL Server using T-SQL procedural code.

You can apply the update operation to a table-like expression using the TABLE (collection_expression) [(+)] clause, and to a specific partition or subpartition using the PARTITION partition_name and SUBPARTITION subpartition_name clauses, respectively. Specific partitions may be affected using the UPDATE statement by including the $PARTITION function to tell SQL Server 2008 which partition to update.

The RETURNING expression retrieves the rows affected by the command, where UPDATE normally only shows the number of rows deleted. The INTO variable clause specifies the variables into which the values returned as a result of the RETURNING clause are stored. There must be a corresponding variable for every expression in the RETURNING clause.

Step Four: Creating Database Objects

In this section, I describe the process of creating interoperable database objects. However, unlike the previous section, I don't detail the exact syntax of every database platform–specific variation. When it comes to database objects such as tables, views, and stored procedures, the differences between databases are manifold and complex. Consequently, I'll describe the most interoperable syntax to create each database object, and follow that up with important notes and details about variations on the various database platforms.

Creating Tables

Manipulating tables is a common activity for database administrators and database programmers. The amount of variation in the CREATE TABLE statement among the various database platforms is quite extreme. The greatest area of variation is in the ways that database platforms implement tables physically on the file system. This variation is because the ANSI SQL standards are abstracted, and don't include any definition for physical implementation of tables. (In fact, because indexes are a means of simply speeding the physical execution of queries against tables, indexes aren't part of the ANSI SQL standard either, believe it or not!) This section details how to create and modify tables.

When writing an interoperable CREATE TABLE statement, use the following syntax:

```
CREATE [TEMPORARY] TABLE table_name
  (column_name data_type attributes [,...] ) |
  [CONSTRAINT constraint_type [constraint_name]
    [constraint_column [,...] ]  [,...] ]
```

The basic CREATE TABLE command describes whether the table is temporary or not, using the keyword TEMPORARY. Next, a comma-delimited series of column definitions follow: the name of the column followed by its datatype—such as INT or VARCHAR(10)—and one or more special attributes.

The two special attributes that are most interoperable are nullability (that is, NULL and NOT NULL), and assigning a DEFAULT value (using the syntax DEFAULT expression, where the expression is an acceptable value for the datatype).

Finally, you may append one or more CONSTRAINT clauses to the CREATE TABLE statement. You use constraints to create PRIMARY KEY and FOREIGN KEY constraints, UNIQUE constraints, CHECK constraints, and to insert DEFAULT values. The constraints are as follows:

* PRIMARY KEY: Declares one or more columns whose values uniquely identify each record in the table
* FOREIGN KEY: Defines one or more columns in the table as referencing columns to a UNIQUE or PRIMARY KEY in another table
* UNIQUE: Declares that the values in one column, or the combination of values in more than one column, must be unique
* CHECK: Compares values inserted into the column that match specific conditions that you define

* DEFAULT: Declares that if no value is provided for the column, then the DEFAULT value will be inserted into the field

The following example shows how to create a table called partners on any major database platform:

```
CREATE TABLE partners
   (prtn_id CHAR(4) NOT NULL,
    prtn_name VARCHAR(40),
    prtn_address1 VARCHAR(40),
    prtn_address2 VARCHAR(40),
    city VARCHAR(20),
    state CHAR(2),
    zip CHAR(5) NOT NULL,
    phone CHAR(12),
    sales_rep INT,
CONSTRAINT pk_prtn_id  PRIMARY KEY (prtn_id),
CONSTRAINT fk_emp_id   FOREIGN KEY (sales_rep)
   REFERENCES employee(emp_id),
CONSTRAINT unq_zip     UNIQUE (zip) );
```

SQL Server

As mentioned earlier, the greatest area of variation among the database platforms is in the physical implementation of a given table. Therefore, avoid the following clauses in the SQL Server implementation of the CREATE TABLE statement when trying to build a statement usable on multiple database platforms:

```
IDENTITY
NOT FOR REPLICATION
ROWGUIDCOL
CLUSTERED | NONCLUSTERED
COLLATE
WITH FILLFACTOR = n
ON filegroup
TEXTIMAGE ON filegroup
```

Although many other platforms support a clause for CLUSTERED or NONCLUSTERED tables, they often mean something quite different from what Microsoft means with its SQL Server tables. In a sense, the term *clustered* means a table that’s physically organized according to an index, while the term *nonclustered* means the table is organized as a heap (that is, data is written physically as it arrives, but in no specific order other than the time that it was created).

MySQL

MySQL is somewhat different from the other database platforms due to its open-source origins. In fact, you can plug several different types of transaction-processing engines into MySQL, which in turn can have a dramatic impact on the type and functionality of tables available to you. For example, only the BDB and InnoDB transaction-processing engines support fully atomic transactions. If you use other types of tables, you won’t be able to execute COMMIT and ROLLBACK statements against transactions operating against the tables in question.

When working on MySQL, avoid the following CREATE TABLE clauses to ensure the highest level of interoperability:

```
IF NOT EXISTS
[MATCH FULL | MATCH PARTIAL | MATCH SIMPLE]
  [ON {DELETE | UPDATE} {RESTRICT | CASCADE | SET NULL | NO ACTION }]
LIKE other_table_name }
 {[TABLESPACE tablespace_name STORAGE DISK] |
  [ENGINE = {ISAM | MyISAM | HEAP | BDB | InnoDB | MERGE | MRG_MyISAM} ] |
  [AUTO_INCREMENT = int] |
  [AVG_ROW_LENGTH = int] |
  [ [DEFAULT] CHARACTER SET charset_name ] |
  [CHECKSUM = {0 | 1}] |
  [ [DEFAULT] COLLATE collation_name ] |
  [COMMENT = "string"] |
  [CONNECTIOIN = 'connection_string'] |
  [DATA DIRECTORY = "path to directory"] |
  [DELAY_KEY_WRITE = {0 | 1}] |
  [INDEX DIRECTORY = " path to directory "] |
  [INSERT_METHOD = {NO | FIRST | LAST}] |
  [KEY_BLOCK_SIZE = int] |
  [MAX_ROWS = int] |
  [MIN_ROWS = int] |
  [PACK_KEYS = {0 | 1}] |
  [PASSWORD = "string"] |
  [ROW_FORMAT= { DEFAULT | DYNAMIC | FIXED | COMPRESSED | REDUNDANT |
COMPACT }]
IGNORE subquery
REPLACE subquery
```

Many of these clauses are only supported by a single specific MySQL table type. For example, all the RAID_xxx clauses are only supported by MyISAM tables.

SQL Server can closely reproduce a number of MySQL clauses in the CREATE TABLE statement, as shown in Table 11-8.

Table 11-8. MySQL CREATE TABLE Clauses That Translate to SQL Server Clauses

MySQL Clause	SQL Server Clause
CREATE TABLE ... LIKE	SELECT...INTO
TYPE = HEAP	PRIMARY KEY NONCLUSTERED
TYPE = {BDB | InnoDB}	<no clause needed>
AUTO_INCREMENT = int	IDENTITY(seed)
DATA DIRECTORY = path(MyISAM table type only)	ON filegroup
INDEX DIRECTORY = path(MyISAM table type only)	ON filegroup
GLOBAL TEMPORARY	TEMPORARY

When translating a SQL Server table to MySQL, be sure to use either the BDB table type or InnoDB table type, depending on which transaction-processing engine is installed with your version of MySQL.

Oracle

Oracle has the most elaborate set of syntax for its CREATE TABLE statement. If you were to compare the vendor documentation of the CREATE TABLE statement, you would find that SQL Server has a little more than 50 pages of documentation, but Oracle has more than 160 pages!

Naturally, any such discussion of the Oracle implementation here will be rather high level and cursory. However, a few patterns in the Oracle implementation of the CREATE TABLE statement bear mentioning. First, Oracle allows an enormous amount of control over the way in which a table is physically implemented on disk. For example, you can define all the physical storage characteristics not only of tables, but also of every type of Large Object (LOB) column, redo logs, and archive logs. You can also do this recursively for each of these structures inside the table.

Oracle also supports several types of tables that aren't supported by SQL Server, including XMLTYPE, object-oriented tables, flashback tables, and more.

When working with Oracle, avoid the following CREATE TABLE clauses to ensure the highest level of interoperability:

```
AS objectype [[NOT] SUBSTITUTABLE AT ALL LEVELS]
   <and subordinate clauses>
OF XMLTYPE
   <and subordinate clauses>
ON COMMIT clause
LOGGING clause
GROUP loggroup
CLUSTER (columnname [,...])
ORGANIZATION clause and subordinate clauses
{ENABLE | DISABLE} ROW MOVEMENT
PARALLEL | NOPARALLEL
NOSORT
COMPRESS | NOCOMPRESS
{ENABLE | DISABLE} [NO]VALIDATE
```

Note that both OBJECT and XMLTYPE tables in Oracle have a number of associated subordinate clauses that apply only to this type of table. Because SQL Server doesn't have an analogous type of table, you should avoid these top-level clauses and all subclauses associated with them.

You can closely reproduce a number of Oracle clauses in the CREATE TABLE statement using SQL Server clauses, as shown in Table 11-9.

Table 11-9. Oracle CREATE TABLE Clauses That Translate to SQL Server Clauses

Oracle Clause	SQL Server Clause
ORGANIZATION INDEX	PRIMARY KEY NONCLUSTERED
ORGANIZATION HEAP	<no clause needed>
PCTFREE	FILLFACTOR

TABLESPACE tablespace_name	ON filegroup
USING INDEX create_index_statement	CONSTRAINT PRIMARY KEY CLUSTERED

Note that although the ORGANIZATION INDEX clause is somewhat like a CLUSTERED INDEX, the syntax is different. Refer to the Oracle documentation for more information about index-organized tables.

Oracle supports the keywords GLOBAL and TEMPORARY.

Creating Indexes

Indexes are contracts that speed query processing by providing (usually) a B-tree of pointers to record locations within a table. Whenever a query has a search argument, such as a WHERE clause, against an indexed column, the query-processing engine can search the much smaller index rather than the entire table, and thereby accelerate query processing. Indexes, as useful as they are, aren’t part of the ANSI SQL standard. That’s because the ANSI standard doesn’t concern itself with questions of physical implementation, and because indexes process directly against the physical layer.

However, there’s an “industry-standard” CREATE INDEX clause that’s highly interoperable across the database platforms:

```
CREATE [UNIQUE] INDEX index_name ON table_name (column_name [, ...])
```

The CREATE INDEX statement first gives a name to the index, then specifies the table and column(s) to which the index applies, in a comma-delimited list. You may alternately declare that the index is UNIQUE, thus preventing the index columns from ever containing a duplicate value. Any insert operation or update operation that attempts to put a duplicate value into a UNIQUE indexed column will fail.

SQL Server

When creating a SQL Server index that’s interoperable with other database platforms, avoid the following clauses:

```
[NON]CLUSTERED
DESC
INCLUDE clause
WITH option
FILESTREAM ON clause

ON filegroup
```

You should avoid all keywords of the WITH option clause.

MySQL

MySQL has two distinctive variations on the standard CREATE INDEX statement. First, you may issue the CREATE FULLTEXT INDEX statement to build a full-text search catalog on a BLOB column. Second, when declaring a column or columns to be indexes, you may also declare a length (in parenthesis) of the column to index. Thus, you might only want to index, say, the first 100 characters of a column that’s VARCHAR(300).

Oracle

Oracle allows you to create indexes on tables, partitioned tables, clusters, index-organized tables, scalar objects of a typed table or cluster, and nested tables. You can specify the physical attributes of an Oracle index, just as you can an Oracle table. You can also specify whether the creation of the index should be parallelized or not. Plus, you can choose from a few types of indexes, including the normal B-tree indexes, partial indexes, function-based indexes, domain indexes, and BITMAP indexes.

Avoid the following Oracle variations of the CREATE INDEX statement for greatest interoperability:

CREATE BITMAP INDEX
Partitioning clauses

```
INDEX TYPE IS index-type
[NO]PARALLEL
[IN]VISIBLE
CLUSTER
[NO]LOGGING
[NO]COMPRESS
[NO]SORT
REVERSE
ONLINE
```

You can partition Oracle tables.

Note Although Oracle has a CLUSTER clause in this statement, it isn't the same thing as a SQL Server clustered index.

You can use the PCTFREE clause to simulate SQL Server's FILLFACTOR and PAD_INDEX clauses.

Creating Views

A *view* is a virtual table that's created from the result set of a predefined query, and is rendered to the user at the time it's queried. In some cases, you can use views not only to retrieve a database, but also insert, update, and delete data from the base tables of the view. The most interoperable form of the CREATE VIEW statement is as follows:

```
CREATE VIEW view_name {[(column [,...])] |
AS
subquery [WITH [CASCADED | LOCAL] CHECK OPTION]
```

SQL Server doesn't support the subquery WITH clause, but this syntax exists on the other major database platforms, and is fairly common. Therefore, you should at least know what this clause means when writing interoperable SQL code or translating from or to SQL Server. The WITH CASCADED CHECK OPTION clause and the WITH LOCAL CHECK OPTION clause tell the DBMS how to behave with nested views. When using the former option, both the main view and any other views it's built upon are checked when an insert, update, or delete operation occurs against the main view. When using the latter option, only the main view is checked against insert, update, and delete operations—even if the main view is built upon other views.

SQL Server

SQL Server allows you to create a basic view with a few extra options. The only statement to avoid when crafting an interoperable CREATE VIEW statement on SQL Server is the WITH option. No other platform supports the view encryption, schemabinding, or view_metadata options.

SQL Server's implementation of the WITH CHECK OPTION is essentially the same as the standard WITH LOCAL CHECK OPTION.

MySQL

MySQL added support for the ANSI standard CREATE VIEW statement in version 5.0. Avoid using MySQL's subclause ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}. Note that MySQL doesn't allow views against temporary tables, while SQL Server does.

Oracle

Oracle supports a rich variety of clauses to the CREATE VIEW statement. Avoid the following clauses when building an interoperable form of the statement:

```
[NO] FORCE
OF type ...
OF XMLTYPE ...
```

The FORCE clause forces the creation of the view regardless of errors, such as missing base tables in the view definition. In this regard, the FORCE clause is somewhat like deferred name resolution in SQL Server.

Also note that Oracle supports the standard WITH CHECK OPTION clause with some optional subclauses, but its functionality is essentially the same as the standard WITH LOCAL CHECK OPTION clause.

Oracle allows you to create a materialized view using the CREATE MATERIALIZED VIEW statement. This is essentially the same as SQL Server's indexed view. Refer to the vendor documentation for details about creating a materialized view.

Creating Triggers

All database platforms covered in this chapter support a certain lowest common denominator for the CREATE TRIGGER statement, as follows:

```
CREATE TRIGGER trigger_name
{BEFORE | AFTER} {DELETE | INSERT | UPDATE }
ON table_name
[FOR EACH {ROW | STATEMENT} ]
BEGIN
  statements
END
```

Using this generic syntax, you can set a trigger to fire before the actual delete, insert, or update operation, or afterwards (though not both) on a given table. The most interoperable triggers specify a trigger time of either BEFORE or AFTER, meaning that the trigger should fire and process its code either before the data manipulation operation takes place or after it takes place.

You may optionally specify whether the trigger logic should fire for each row of a transaction, if the transaction spans many records, or if it should fire for each statement in the transaction, firing only once for a statement that might span many records. Finally, a BEGIN...END block encases the logic that will substitute for the actual DELETE, INSERT, or UPDATE statement.

Also note that triggers, according to the ANSI SQL standard, have programmatic access to two important pseudotables: OLD and NEW. The OLD pseudotable contains all the data or records before the data manipulation operation that fired the trigger occurs. Conversely, the NEW pseudotable contains all the data or records after the data manipulation operation fired against the table. These pseudotables have the exact same schema of the base table that the trigger is operating against. Thus, you can use these pseudotables to see what the data is before and after the trigger fires, to manipulate the data conditionally as you see fit.

SQL Server

SQL Server supports the syntax for the interoperable CREATE TRIGGER statement with some minor variations. SQL Server also supports triggers for purposes other than DML statements, such as DDL triggers and audit triggers. Avoid these variations when writing interoperable code for other database platforms. SQL Server triggers always operate in FOR EACH STATEMENT mode. First, SQL Server allows you to create a single time, but multiple events trigger in a single statement. Thus, the following code is possible:

```
CREATE TRIGGER my_trigger ON my_table
AFTER DELETE, INSERT, UPDATE
AS  BEGIN
  statements
END
```

Note SQL Server uses the keyword FOR instead of BEFORE.

On the other hand, you should avoid the following clauses altogether when writing an interoperable CREATE TRIGGER statement on SQL Server:

```
INSTEAD OF
WITH ENCRYPTION
WITH APPEND
NOT FOR REPLICATION
IF UPDATE...
```

SQL Server supports an additional trigger activation event called INSTEAD OF. Rather than firing before or after the data-manipulation operation that invoked the trigger, INSTEAD OF triggers completely substitute their processing for that of the invoking data-manipulation operation.

SQL Server allows you to create many triggers for a given time and event for a single table or view. Thus, you could have three FOR UPDATE triggers and two AFTER DELETE triggers. You may only have one INSTEAD OF trigger per DML operation per table. You can set the first and last trigger to fire using the sp_settriggerorder system stored procedure. Otherwise, the order in which the triggers fire is undefined.

As the odd man out, SQL Server refers to the trigger pseudotables as DELETED and INSERTED, rather than OLD and NEW.

SQL Server 2008 also supports DDL trigger events on actions such as GRANT or CREATE. This is a useful addition for users who might want to build a trigger-based auditing system for their application that encompasses not only data manipulation activities but also database administration activities.

MySQL

MySQL supports the interoperable version of the CREATE TRIGGER statement with the exception that it only supports the FOR EACH ROW clause and not the FOR EACH STATEMENT clause. MySQL calls the two trigger pseudotables OLD and NEW, as does the standard.

MySQL currently allows only one trigger of each time and action (for example, a BEFORE INSERT trigger or an AFTER UPDATE trigger) on a given table.

When writing interoperable MySQL triggers, avoid the *DEFINER* clause.

Oracle

Oracle supports the interoperable form of the CREATE TRIGGER statement with the added trigger time of INSTEAD OF, as does SQL Server. Oracle also calls the trigger pseudotables OLD and NEW, though you can rename them using the following syntax:

```
FOLLOWS clause

REFERENCING [OLD [AS] old_table_name] [NEW [AS] new_table_name]
```

When building an interoperable CREATE TRIGGER statement on Oracle, avoid the [OR] clause, the PARENT clause, and the WHEN condition clause.

Oracle allows a number of object events to fire whenever Oracle encounters a specific keyword such as DROP, GRANT, or TRUNCATE. You can easily translate these into SQL Server 2008 using DDL triggers. You'll have to do some fancy programming via a stored procedure if you want to translate this behavior to SQL Server 2000 or earlier.

Creating Procedures and Functions

Most of the database platforms in this chapter support the ability to create stored procedures and user-defined functions. Because the ANSI SQL standard only approached the issue of procedural extensions in the 2003 standards, each of the database platforms has had to address issues of providing adequate procedural language capability at various points in their history by creating their own widely divergent SQL dialects.

Note The ANSI standard calls both procedures and user-defined functions *routines*. Other types of procedural code such as types and methods are also grouped into the wide term of "routines."

SQL Server (and its forbearer Sybase) uses the T-SQL dialect. SQL Server 2008 also allows .NET languages such as C# to be compiled into stored procedures via the CLR interface. Oracle uses the venerable and powerful PL/SQL dialect, as well as Java. MySQL allows only SQL stored procedures and user-defined functions.

Note SQL Server allows stored procedures to pass out result sets, such as the result from an entire SELECT statement. Most other database platforms, most notably Oracle, don't allow stored procedures to do this. Instead, you must design your stored procedures to dump any important result sets to temporary holding tables where you can later pull out and process those result sets according to the needs of your application.

A full description of each of these SQL dialects is far beyond the scope of this chapter. However, all the platforms in this chapter support a rudimentary syntax for the CREATE PROCEDURE and CREATE FUNCTION statement.

For CREATE PROCEDURE, use the following code:

```
CREATE PROCEDURE routine_name
  ( [{[IN | OUT | INOUT]} [parameter_name] data type [,...] ])
BEGIN
  routine_body
END
```

For CREATE FUNCTION, use the following code:

```
CREATE FUNCTION routine_name ([parameter[,...]])
  ( [{[IN | OUT | INOUT]} [parameter_name] data type [,...] ])
  RETURNS scalar_value
BEGIN
  routine_body
END
```

The parameter_name uniquely identifies one or more parameters for the routine. The parameter can carry a value IN to the routine, OUT from the routine, or both into and out of (INOUT) the routine. Functions must pass out a single scalar value, though stored procedures may pass out other sorts of values.

The routine_body should contain only SQL statements or procedure calls for maximum interoperability. If the routine_body contains any sort of code for a local SQL dialect, you can expect to have a good deal of work ahead in translating the procedural code.

Note that only SQL Server supports options such as ENCRYPTION and SCHEMABINDING.

Best Practices

A constant theme of this chapter is: *the closer you can stick to the ANSI standard the more interoperable you will be.* This is just a rule of thumb, because SQL Server isn't totally compliant with the ANSI standards itself, but it's a strong rule. Here are some other broader best practices that, while not strictly expansions of the ANSI standard best practice rule, are at least corollaries:

* When using datatypes, be sure to stick to the ANSI standard datatypes. In particular, when designing for interoperability, avoid the following datatypes: CURSOR, IMAGE, SQL_VARIANT, TABLE, VARCHAR(max), and TEXT. DATETIME is problematic because the SQL Server implementation is so different from the ANSI standard and the implementation of the datatype.
* Avoid quoted, bracketed, or otherwise delimited identifiers.
* When working with DELETE, INSERT, SELECT, and UPDATE statements, avoid hints, non-ANSI function calls, the WITH clause, and the ANSI ONLY clause.
* Don't use the second FROM clause to implement JOINS in the SQL Server DELETE and UPDATE statements.
* Avoid the SQL Server INSERT...EXECUTE statement.
* Remember that the SQL Server SELECT statement has many clauses that aren't interoperable, including the INTO clause, CUBE and ROLLUP clause, COMPUTE clause, and the FOR XML clauses.
* Avoid PARTITIONs in your database schema unless you only plan to interoperate with SQL Server and Oracle.
* Remember that clustered indexes on SQL Server are different from a "clustered" index on other platforms.
* Recognize that database routines (procedures, functions, and so on) are predicated on the database platform's SQL dialect (T-SQL for SQL Server, PL/SQL for Oracle, and so forth). Do not plan to build interoperable database routines. They almost always require a rather extended migration effort.

With these best practices alone, you'll be cranking out SQL code and SQL Server databases that are far more interoperable than before, and require little work to make fully interoperable.

Summary

Looking back over this chapter, you can see that there are a number of issues when writing code that's interoperable between SQL Server and the other major database platforms. Whether you want to move code from SQL Server or to SQL Server from another database platform, you'll likely have to change some of the code so that it can run properly. There are a number of areas where you have to be careful, from naming identifiers and choosing datatypes to writing transportable SELECT and CREATE TABLE statements. This chapter helped ease all those requirements by informing you how SQL Server implements each statement, and comparing that to how the other database platforms accomplish the same thing. Thus, you’re able to write much more interoperable and transportable code from the start.