



## Manipulating Large Data Sets

## CERTIFICATION OBJECTIVES

### 13.01 Describe the Features of Multitable INSERTs

### 13.02 Merge Rows into a Table

✓ Two-Minute Drill

## **O&A** Self Test

This chapter looks at features and operations that are useful for working with large groups of data. We cover two operations in particular. The first consists of additional features to the INSERT statement that enable you to use one INSERT statement to add multiple rows of data to a given table or to several tables, with and without conditional logic. Later in the chapter we'll cover the SQL statement MERGE, which is a powerful statement that enables you to combine the features of multiple Data Manipulation Language (DML) statements into a single SQL statement.

CERTIFICATION OBJECTIVE 13.01

## DESCRIBE THE FEATURES OF MULTITABLE INSERTS

The multitable INSERT statement is a variation on the INSERT statement syntax you've already seen. A multitable INSERT statement repeats the INTO clause of the INSERT statement to insert data into more than one table. Each INTO clause applies to just one table, but by repeating the INTO clause, you can add data to multiple tables. The multitable INSERT must have a subquery to select rows for inserting.

Multitable INSERT statements can accomplish a variety of tasks, including the following:

- Query data from one table and insert the data into multiple tables with conditional logic, such as transforming data into a series of archive tables.
- Exchange data between two similar systems of different requirements—perhaps between a transaction-based application and a data warehouse optimized for analysis.
- Support logical archiving at any level of detail with logical decision points embedded in the INSERT statements.
- Integrate complex queries with GROUP BY, HAVING, set operators, and more, all while moving any number of rows dynamically, distributing output into multiple data targets, and programming logical decision points to control data distribution.
- Transform data that is stored in rows and levels into a cross-tabulation output, the type you would typically see in a spreadsheet application.

There are two general types of multitable INSERT statements: unconditional and conditional.

- Unconditional multitable INSERT statements process each of the INSERT statement's one or more INTO clauses, without condition, for all rows returned by the subquery.

Let's look at the overall syntax for the multitable INSERT statement. First, we'll examine an unconditional multitable INSERT statement. The syntax repeats the INTO statement from one to many times as required.

The unconditional multitable INSERT statement syntax just shown assumes the following:

- The conditional multitable INSERT statement syntax is similar but adds the WHEN condition, like this:

For each row returned by the subquery, each WHEN condition is evaluated and determined to be either true or false. If true, then the WHEN condition's associated set of one or more INTO clauses is executed; otherwise, processing skips over the INTO clauses to the next WHEN condition. If none of the WHEN conditions evaluates to true, the ELSE clause is processed, and its associated set of one or more INTO clauses is executed.

- The *option* is one of two keywords: ALL or FIRST.
- ALL is the default and may be omitted.
- FIRST is the alternative keyword; it indicates that the only set of INTO clauses that will execute are those that follow the first WHEN clause that evaluates to true.
- You can include multiple WHEN conditions.
- Each WHEN condition is followed by one or more INTO clauses.
- Each INTO may have its own VALUES clause; if omitted, the subquery's select list must match the number and data types of the INTO table's columns.
- Each *expression* evaluates to true or false and should involve one or more columns from the subquery.
- The *tab* and *col\_list* are the components of the INSERT statement that will execute if the WHEN expression evaluates to true.
- The optional ELSE . . . INTO clause, if included, must be last.
- The *subquery* is required and must be a valid SELECT statement.

The multitable INSERT statement always uses a subquery. As you know, a subquery may return anywhere from zero to many rows. For each row returned by a subquery, processing does a pass through the set of WHILE ... INTO clauses. But the way it processes the WHILE ... INTO clauses differs based on whether the keyword ALL or FIRST is used.

clauses) will not be executed. All the WHEN conditions are evaluated if the keyword ALL is specified at the beginning of the multitable INSERT statement.

On the other hand, if the keyword FIRST is used, then for each row returned by the subquery, WHEN conditions are evaluated until the first true condition is encountered. As soon as a WHEN condition is determined to be true, the corresponding set of one or more INTO clauses that follows the WHEN will be executed. Processing will then skip over the remaining WHEN conditions for that row of the subquery.

In either situation—INSERT FIRST or INSERT ALL—if no WHEN condition was found to be true and if the optional ELSE clause is present, then the ELSE clause's INTO clause will be executed for the row. Then processing moves on to the next row returned by the subquery.

Note that for the conditional multitable INSERT statement—which is to say any multitable INSERT with a WHEN condition—ALL is the default keyword. If no WHEN condition is used, then the multitable INSERT is unconditional, and the ALL keyword must be present.

In other words, you may not omit the keyword in an unconditional multitable INSERT, like this:

```
INSERT
  INTO ... VALUES ...
  INTO ... VALUES ...
subquery;
```

The preceding statement shows incorrect syntax since it omits the ALL option and yet has no WHEN condition. Therefore, it is syntactically incorrect. However, you may do something like this:

```
INSERT ALL
  INTO ... VALUES ...
  INTO ... VALUES ...
subquery;
```

The preceding unconditional multitable INSERT correctly shows the ALL keyword.

The conditional multitable INSERT allows you to omit the keyword, like this:

```
INSERT
  WHEN ... THEN
    INTO ... VALUES ...
  WHEN ... THEN
    INTO ... VALUES ...
subquery;
```



*Multitable INSERT statements require a subquery.*

The default keyword is ALL. In all forms of the multitable INSERT, the subquery is required; it is not optional. And as is always the case with any INSERT that uses a subquery, the INSERT statement will execute once for each row returned by the subquery.

Note that if any one INTO clause fails with an execution error for any one row returned by the subquery, then the entire statement fails for all rows of the subquery, and no data change results.

Use the Following Types of Multitable INSERTS:  
Unconditional and Conditional

Let's look at some examples of the multitable INSERT statement in action. First up are unconditional multitable INSERTS, followed by conditional examples. Then we'll consider the unique and useful approach to perform the equivalent of a spreadsheet pivot, a common use of the multitable INSERT.

Unconditional

As we discussed, the unconditional multitable INSERT statement omits conditional logic, in other words, the WHEN clause. For example, consider the CRUISE\_ORDERS table, as shown in Figure 13-1.

FIGURE 13-1 Diagram for the CRUISE\_ORDERS table

CRUISE_ORDERS	
P * CRUISE_ORDER_ID	NUMBER
ORDER_DATE	DATE
CRUISE_CUSTOMER_ID	NUMBER
SHIP_ID	NUMBER (7)
PK_CRUISE_ORDER_ID	

Here is an example of a valid SQL statement that queries the CRUISE\_ORDERS table and inserts the output into each of our three archive tables (line numbers added):

```

01  INSERT ALL
02      INTO CO_2018 (CRUISE_ORDER_ID, ORDER_DATE,
03                   CRUISE_CUSTOMER_ID, SHIP_ID)
04  VALUES (CRUISE_ORDER_ID, ORDER_DATE,
05           CRUISE_CUSTOMER_ID, SHIP_ID)
06  INTO CO_ELCARO (CRUISE_ORDER_ID, ORDER_DATE,
07                 CRUISE_CUSTOMER_ID, SHIP_ID)
08  VALUES (CRUISE_ORDER_ID, ORDER_DATE,
09           CRUISE_CUSTOMER_ID, SHIP_ID)
10  INTO CO_ARCHIVED (CRUISE_ORDER_ID, ORDER_DATE,
11                   CRUISE_CUSTOMER_ID, SHIP_ID)
12  VALUES (CRUISE_ORDER_ID, ORDER_DATE,
13           CRUISE_CUSTOMER_ID, SHIP_ID)
14  SELECT CRUISE_ORDER_ID, ORDER_DATE, CRUISE_CUSTOMER_ID, SHIP_ID
15  FROM CRUISE_ORDER;

```

Note that we have three INTO clauses here. If the subquery returns, for example, three rows, then the result of this INSERT statement will be to insert nine rows: three into the CO\_2018 table (line 2), three into the CO\_ELCARO table (line 6), and three into the CO\_ARCHIVED table (line 10).

As we see in the preceding example, the unconditional INSERT statement uses the keyword ALL (line 1), followed by one or more INTO clauses (lines 2, 6, and 10), each of which specifies a table and the columns into which we are inserting data, followed by the VALUES list.

The VALUES list can specify expressions found in the subquery's select list. In our example, in line 4 we specify CRUISE\_ORDER\_ID as the first expression in the VALUES list to be inserted into the CO\_2018 table. This corresponds to the CRUISE\_ORDER\_ID column in the subquery select list in line 14. The other VALUES lists that refer to CRUISE\_ORDER\_ID (line 8 and line 12) are specifying that same column. Each VALUES list in a multitable INSERT can specify any column names or expressions that are in the subquery select list.

On the other hand, the column references within each INTO list (each starting at lines 2, 6, and 10) specify the columns of the tables named for the INTO clause. In our example, line 2 names the CO\_2018 table, and the INTO list that follows on line 2 and line 3 specifies columns in the CO\_2018 table.

You'll recall that in a standard INSERT statement, the list of values in the VALUES expression list must match in number and in data type (or be able to be automatically converted to a matching data type) with the columns specified in the INTO clause. The same is true here for each pair of INTO and VALUES lists.

Each VALUES expression list may use any complex expression in specifying the value to be inserted into its corresponding table and column. Here's an example:

```

01 INSERT ALL
02 INTO CO_2018 (CRUISE_ORDER_ID, ORDER_DATE,
03 CRUISE_CUSTOMER_ID, SHIP_ID)
04 VALUES (CRUISE_ORDER_ID, ORDER_DATE, 14, 1)
05 INTO CO_ELCARO (CRUISE_ORDER_ID, ORDER_DATE,
06 CRUISE_CUSTOMER_ID, SHIP_ID)
07 VALUES (CRUISE_ORDER_ID, ORDER_DATE+30, 15, 1)
08 INTO CO_ARCHIVED (CRUISE_ORDER_ID, ORDER_DATE,
09 CRUISE_CUSTOMER_ID, SHIP_ID)
10 VALUES (CRUISE_ORDER_ID, ORDER_DATE,
11 CRUISE_CUSTOMER_ID, SHIP_ID)
12 SELECT CRUISE_ORDER_ID, ORDER_DATE, CRUISE_CUSTOMER_ID, SHIP_ID
13 FROM CRUISE_ORDERS;

```

In this example, we are choosing to insert some values that are different from what the subquery is returning. For the CO\_2018 table, in lines 2 through 4, we are defining the ORDER\_DATE value for all rows to be SYSDATE, the CRUISE\_CUSTOMER\_ID value to be the literal value of 14, and the SHIP\_ID value to be a literal value of 1. For the CO\_ELCARO table, in lines 5 through 7, we are giving each row an ORDER\_DATE value that is 30 days beyond the incoming value in the subquery, and we're assigning the number 15 to each CRUISE\_CUSTOMER\_ID and assigning 1 to each SHIP\_ID. For the CO\_ARCHIVED table, in lines 8 through 11, we are choosing to pass through values from the subquery unchanged.

As the example shows, the VALUES list can specify column names and expressions from the subquery's select list, but may also define any valid SQL expression. The INTO column list must specify columns in the table into which the INTO statement is inserting data.

If the VALUES list is omitted, the columns of the subquery become the de facto VALUES list and therefore must match the columns of the corresponding INTO clause. By “match,” we mean that they must match in number and in data type, or be of such data types that an automatic data type conversion may be performed.

If there is no column list in the INTO clause, the subquery's select list must match the columns in the table of the INTO clause.

## Conditional

Conditional multitable INSERT statements use conditional logic to determine which INTO clause or clauses to process. Each row that is returned by the subquery is processed through a series of one or more WHEN conditions. Each WHEN condition is followed by a set of one or more INTO clauses.

For each row returned by the subquery, each WHEN condition is evaluated to be either true or false. If true, the following set of one or more INTO clauses is executed. If false, the set of one or more INTO clauses is skipped, and the next WHEN condition is evaluated.

An EISE clause may optionally be included in the conditional multitable

skipped for any given row, then the ELSE clause's INTO will be processed. Otherwise, it will be skipped for that row.

Each row returned by the subquery is processed according to these rules we have just reviewed.

Let's look again at our table INVOICES and the archive table INVOICES\_ARCHIVED, in which we stored invoice records that are more than a year old. See Figure 13-1 for the INVOICES table, and see Figure 13-2 for the INVOICES\_ARCHIVED table.

FIGURE 13-2 Diagram for the INVOICES\_ARCHIVED table

INVOICES_ARCHIVED	
INVOICE_ID	NUMBER
INVOICE_DATE	DATE
ACCOUNT_NUMBER	VARCHAR2 (80 BYTE)
TERMS_OF_DISCOUNT	VARCHAR2 (20 BYTE)
VENDOR_ID	NUMBER
TOTAL_PRICE	NUMBER (8,2)
SHIPPING_DATE	DATE

Let's say our organization is engaged in a merger and we are tasked with the job of integrating data from another application. The newly acquired company has provided us with the table WO\_INV, as shown in Figure 13-3.

FIGURE 13-3 Diagram for the WO\_INV table

WO_INV	
P * INV_NO	NUMBER (11)
DATE_ENTERED	DATE
DATE_SHIPPED	DATE
CUST_ACCT	VARCHAR2 (30 BYTE)
IX_SYS_C0012962	
SYS_C0012962	

We need to create an INSERT statement that will

- Pull data from the WO\_INV table
- Insert WO\_INV's invoice information from within the past year into our INVOICES table
- Insert WO\_INV's invoice information that is more than a year old into our INVOICES\_ARCHIVED table

It's a perfect task for a conditional multitable INSERT statement, as follows:

```
01 INSERT FIRST
02 WHEN (DATE_SHIPPED < (ADD_MONTHS(SYSDATE,-12))) THEN
03   INTO INVOICES_ARCHIVED (INVOICE_ID, INVOICE_DATE,
04                           SHIPPING_DATE, ACCOUNT_NUMBER)
05   VALUES (INV_NO, DATE_ENTERED, DATE_SHIPPED, CUST_ACCT)
06 ELSE
07   INTO INVOICES (INVOICE_ID, INVOICE_DATE,
08                 SHIPPING_DATE, ACCOUNT_NUMBER)
09   VALUES (INV_NO, DATE_ENTERED, DATE_SHIPPED, CUST_ACCT)
10 SELECT INV_NO, DATE_ENTERED, DATE_SHIPPED, CUST_ACCT
11 FROM   WO_INV;
```

In this statement, we see the following:

- A subquery on lines 10 through 11. Note the subquery includes a column DATE\_SHIPPED.
- Line 2 compares the DATE\_SHIPPED value in a WHEN condition.
- If line 2 evaluates to true for a given row from the subquery, the INSERT statement will take that row's data and insert it into the INVOICES\_ARCHIVED table, as specified on line 3. The columns in the INVOICES\_ARCHIVED table are specified in lines 3 and 4.
- Line 5 defines the values from the subquery that will be inserted if the WHEN clause on line 2 is true. For example, the subquery's column INV\_NO (line 5) will be inserted into the target table's column INVOICE\_ID (line 3).
- Line 6 is an ELSE clause that will execute for each row that does not satisfy the WHEN condition in line 2.

In the example we just reviewed, there was one WHEN condition and one ELSE condition. Let's look at an example with multiple WHEN conditions. Let's say you had three archive tables, named INVOICES\_THRU\_2019, INVOICES\_THRU\_2018, and INVOICES\_THRU\_2017, and wanted to insert rows from the incoming table into each archived table based on the year of the DATE\_SHIPPED value. Note that each table is not mutually exclusive; for example, the INVOICES\_THRU\_2019 table will contain invoices from 2019, 2018, and 2017, as well as earlier. One row returned by the subquery might be inserted into all three tables.

To accomplish this task, you could use the following INSERT statement:

```
01 INSERT
02 WHEN (TO_CHAR(DATE_SHIPPED,'RRRR') <= '2019') THEN
03   INTO INVOICES_THRU_2019 (INVOICE_ID, INVOICE_DATE,
04                           SHIPPING_DATE, ACCOUNT_NUMBER)
05   VALUES (INV_NO, DATE_ENTERED, DATE_SHIPPED, CUST_ACCT)
06 WHEN (TO_CHAR(DATE_SHIPPED,'RRRR') <= '2018') THEN
07   INTO INVOICES_THRU_2018 (INVOICE_ID, INVOICE_DATE,
08                           SHIPPING_DATE, ACCOUNT_NUMBER)
09   VALUES (INV_NO, DATE_ENTERED, DATE_SHIPPED, CUST_ACCT)
10 WHEN (TO_CHAR(DATE_SHIPPED,'RRRR') <= '2017') THEN
11   INTO INVOICES_THRU_2017 (INVOICE_ID, INVOICE_DATE,
12                           SHIPPING_DATE, ACCOUNT_NUMBER)
```

Notice that there is no keyword FIRST or ALL in this example. Therefore, the statement will default to ALL. Since there are three WHEN conditions, each with an associated INTO clause, each and every WHEN condition that evaluates to true will execute. Also, this example omits the ELSE clause, so if any row from the subquery does not satisfy a WHEN condition, then no action will be taken for that particular row returned by the subquery.

After any WHEN condition, you may include more than one INTO clause. For example, let's say we have a table INVOICES\_CLOSED that takes any invoice rows that shipped prior to 2018. We might modify our example like this:

```
01  INSERT
02  WHEN (TO_CHAR (DATE_SHIPPED, 'RRRR') <= '2019') THEN
03  INTO INVOICES_THRU_2019 (INVOICE_ID, INVOICE_DATE,
04  SHIPPING_DATE, ACCOUNT_NUMBER)
05  VALUES (INV_NO, DATE_ENTERED, DATE_SHIPPED, CUST_ACCT)
06  WHEN (TO_CHAR (DATE_SHIPPED, 'RRRR') <= '2018') THEN
07  INTO INVOICES_THRU_2018 (INVOICE_ID, INVOICE_DATE,
08  SHIPPING_DATE, ACCOUNT_NUMBER)
09  VALUES (INV_NO, DATE_ENTERED, DATE_SHIPPED, CUST_ACCT)
10  INTO INVOICES_CLOSED (INVOICE_ID, INVOICE_DATE,
11  SHIPPING_DATE, ACCOUNT_NUMBER)
12  VALUES (INV_NO, DATE_ENTERED, DATE_SHIPPED, CUST_ACCT)
13  WHEN (TO_CHAR (DATE_SHIPPED, 'RRRR') <= '2017') THEN
14  INTO INVOICES_THRU_2017 (INVOICE_ID, INVOICE_DATE,
15  SHIPPING_DATE, ACCOUNT_NUMBER)
16  VALUES (INV_NO, DATE_ENTERED, DATE_SHIPPED, CUST_ACCT)
17  SELECT INV_NO, DATE_ENTERED, DATE_SHIPPED, CUST_ACCT
18  FROM   WO_INV;
```

Note the new INTO clause, lines 10 through 12. This INTO is subject to the WHEN condition in line 6. In other words, if DATE\_SHIPPED is in the year 2018 or before, the INSERT statement will add the candidate row to both the INVOICES\_THRU\_2018 table and the INVOICES\_CLOSED table. One WHEN condition is the gateway to both INTO clauses.

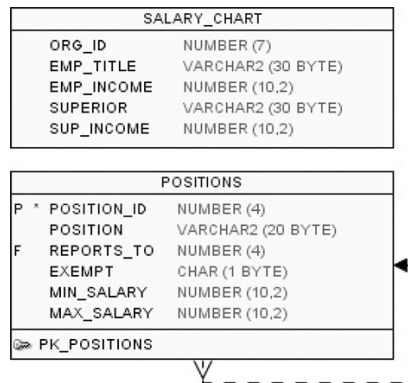
What this example shows us is that any WHEN condition can have multiple INTO clauses that follow it. If the WHEN condition evaluates to true, all of its INTO clauses will execute. If the WHEN condition evaluates to false, execution will skip over the INTO clauses and move on directly to either the next WHEN condition, an ELSE if it is present, or the next row in the subquery.

The INSERT ALL will evaluate each and every WHEN condition and process all INTO clauses for all WHEN conditions that evaluate to true. Therefore, the INSERT ALL may result in a single row being added to more than one table.

The INSERT FIRST will evaluate every WHEN condition until one of them evaluates to true. It will then process that WHEN condition's INTO and skip the remaining WHEN conditions. The INSERT FIRST will process only zero or one WHEN condition; however, it may also result in a single row being added to more than one table, but only if the first true WHEN condition has more than one INTO clause.

Table aliases in the subquery of a multitable INSERT are not recognized in the rest of the INSERT. For example, you can't reference them from within a WHEN condition or INTO statement. If a subquery's column reference depends on a table alias, be sure to use a column alias for the column and then reference the column alias if you want to reference that column from elsewhere within the INSERT statement—outside of the subquery. For example, see [Figure 13-4](#).

**FIGURE 13-4** Diagram of the POSITIONS and SALARY\_CHART tables



In a query that queries rows from the POSITIONS table and conditionally inserts them into the SALARY\_CHART table, we cannot use this query:

```
01  INSERT
02  WHEN (B.MAX_SALARY - A.MAX_SALARY < 10000) THEN
03  INTO SALARY_CHART (EMP_TITLE, SUPERIOR,
04  EMP_INCOME, SUP_INCOME)
05  VALUES
06  (A.POSITION, B.POSITION,
07  A.MAX_SALARY, B.MAX_SALARY)
08  SELECT A.POSITION,
09  B.POSITION,
10  A.MAX_SALARY,
11  B.MAX_SALARY
12  FROM   POSITIONS A JOIN POSITIONS B
13  ON     A.REPORTS_TO = B.POSITION_ID
14  WHERE  A.MAX_SALARY > 100000;
```

This statement will not work. Here is the result:

```
Error at Command Line:6 Column:35
Error report:
SQL Error: ORA-00904: "B"."MAX_SALARY": invalid identifier
ORA-00904: "B"."MAX_SALARY": invalid identifier
```

the subquery but are not recognized beyond the subquery. In other words, the attempts to reference each table alias from within the WHEN condition or VALUES clause are invalid.

So, what do we do? The solution is to specify a column alias to any column names within the subquery that use a table alias and then reference the column aliases from the rest of the conditional INSERT statement, like we do in the following lines 5 and 6:

```

01 INSERT
02 WHEN (BOSS_SALARY - EMPLOYEE_SALARY < 10000) THEN
03 INTO SALARY_CHART (EMP_TITLE, SUPERIOR, EMP_INCOME, SUP_INCOME)
04 VALUES (EMPLOYEE, BOSS, EMPLOYEE_SALARY, BOSS_SALARY)
05 SELECT A.POSITION EMPLOYEE,
06        B.POSITION BOSS,
07        A.MAX_SALARY EMPLOYEE_SALARY,
08        B.MAX_SALARY BOSS_SALARY
09 FROM POSITIONS A JOIN POSITIONS B
10 ON A.REPORTS_TO = B.POSITION_ID
11 WHERE A.MAX_SALARY > 10000;

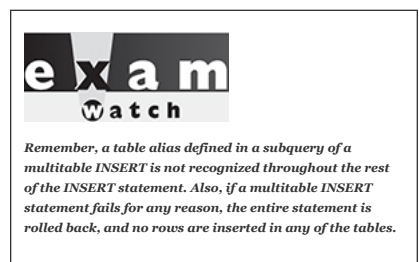
```

Note that this version has done more than is required; it applies column aliases to each column in the subquery and then references those column aliases from the WHEN and VALUES clauses. We only needed column aliases on A.POSITION and B.POSITION in lines 5 and 6, so we can reference the column aliases in line 4. Either way, this version of the conditional INSERT is syntactically correct.

You cannot execute a multitable INSERT on a view; it can be used only with a table.

Sequence generators do not behave consistently in a multitable INSERT statement. If you try to use a sequence generator within the subquery, you'll get a syntax error. If you try to include one within the expression list of the INTO statement, you may or may not get the functionality you want—the NEXTVAL function will not advance as you might expect. The reason is that a multitable insert is treated as a single SQL statement. Therefore, if you reference NEXTVAL with a sequence generator, Oracle's documentation warns that NEXTVAL will be incremented once in accordance with the sequence generator's parameters and stay that way for the duration of a pass through the multitable insert. In other words, a conditional INSERT with a single INTO, one that invokes a single sequence generator once with a NEXTVAL, will increment the sequence once for each row returned by the subquery—regardless of whether the WHEN condition is true. Consider this example:

```
01  INSERT
02      WHEN (TO_CHAR (DATE_ENTERED, 'RRRR') <= '2019') THEN
03          INTO INVOICES_ARCHIVED (INVOICE_ID, INVOICE_DATE)
04          VALUES (SEQ_INV_NUM.NEXTVAL, DATE_ENTERED)
05  SELECT INV_NO, DATE_ENTERED FROM WO INV;
```



The sequence generator in line 4 will increment for each row returned by the subquery, regardless of whether the WHEN condition is true. For this example, assume the sequence generator has just been created and has never been used and that it has the default settings of an initial value of 1 and an increment of 1. Given that, if the subquery returns ten rows and if, for instance, the final row alone causes the WHEN condition in line 2 to be true, then the one row inserted into the INVOICES\_ARCHIVED table will be assigned a value of 10 for the INVOICE\_ID column.

If this statement contained additional calls to the same sequence generator in additional INTO clauses, they would not cause the sequence generator to increment. The sequence generator increments with each row of the subquery returned—no more, no less—regardless of additional calls to NEXTVAL.

Oracle's documentation warns that "you cannot specify a sequence in any part of a multitable insert statement." The only place you'll get the syntax error is in the subquery, but know that attempts to invoke a sequence generator from the WHEN or INTO clause of the INSERT may produce undesirable results.

## Using Multitable INSERT to Perform a Pivot

You can use a conditional multitable INSERT statement to transform data from a spreadsheet structure to a rows-and-columns structure. This section describes the technique.

First, let's start with the following data listing:

ROOM_TYPE	OCEAN	BALCONY	NO_WINDOW
ROYAL	1745	1635	
SKYLOFT	722	722	
PRESIDENTIAL	1142	1142	1142
LARGE	225		211
STANDARD	217	554	586

This is the sort of data you might find in a typical spreadsheet display. Let's say this spreadsheet has been stored in an external table. [Figure 13-5](#) shows the table's structure.

**FIGURE 13-5** Diagram of the SHIP CABIN GRID table

SHIP_CABIN_GRID	
ROOM_TYPE	VARCHAR2 (20 BYTE)
OCEAN	NUMBER
BALCONY	NUMBER
NO_WINDOW	NUMBER

Next, you're given the task of moving this data into the table, as shown in Figure 13-6.

FIGURE 13-6 Diagram of the SHIP\_CABIN\_STATISTICS table

SHIP_CABIN_STATISTICS	
SC_ID	NUMBER (7)
ROOM_TYPE	VARCHAR2 (20 BYTE)
WINDOW_TYPE	VARCHAR2 (10 BYTE)
SQ_FT	NUMBER (8)

This isn't a straightforward row-for-row insert with a subquery. This data must be transformed so that each column from the spreadsheet is transformed into an individual row in the new table.

The following query will accomplish the task:

```
01 INSERT ALL
02   WHEN OCEAN IS NOT NULL THEN
03     INTO SHIP_CABIN_STATISTICS (ROOM_TYPE, WINDOW_TYPE, SQ_FT)
04   VALUES (ROOM_TYPE, 'OCEAN', OCEAN)
05   WHEN BALCONY IS NOT NULL THEN
06     INTO SHIP_CABIN_STATISTICS (ROOM_TYPE, WINDOW_TYPE, SQ_FT)
07   VALUES (ROOM_TYPE, 'BALCONY', BALCONY)
08   WHEN NO_WINDOW IS NOT NULL THEN
09     INTO SHIP_CABIN_STATISTICS (ROOM_TYPE, WINDOW_TYPE, SQ_FT)
10   VALUES (ROOM_TYPE, 'NO WINDOW', NO_WINDOW)
11 SELECT ROWNUM RN, ROOM_TYPE, OCEAN, BALCONY, NO_WINDOW
12 FROM SHIP_CABIN_GRID;
```

Note how each row of the subquery is considered three times. For a given row returned by the subquery, each of three columns of the row is individually considered. If any one of the three columns OCEAN, BALCONY, and NO\_WINDOW is not NULL, then a row is inserted into the target table. It's possible that some individual rows returned by the subquery will result in three new rows being added to the target table SHIP\_CABIN\_STATISTICS.

Let's take a look at the results:

```
SELECT ROOM_TYPE, WINDOW_TYPE, SQ_FT
FROM SHIP_CABIN_STATISTICS
ORDER BY ROOM_TYPE, WINDOW_TYPE;
```

ROOM_TYPE	WINDOW_TYPE	SQ_FT
LARGE	NO WINDOW	211
LARGE	OCEAN	225
PRESIDENTIAL	BALCONY	1142
PRESIDENTIAL	NO WINDOW	1142
PRESIDENTIAL	OCEAN	1142
ROYAL	BALCONY	1635
ROYAL	OCEAN	1745
SKYLOFT	BALCONY	722
SKYLOFT	OCEAN	722
STANDARD	BALCONY	554
STANDARD	NO WINDOW	586
STANDARD	OCEAN	217

In this example, the conditional multitable INSERT transformed incoming data from a spreadsheet summary style into a row-by-row structure, all within a single SQL statement. In this way, the conditional multitable INSERT statement "pivots" the data by changing columns into rows.



**Note that this pivot technique is different from SQL operations that use the keyword PIVOT or UNPIVOT. What we've described here is a technique that uses the conditional multitable INSERT to pivot data. The keyword PIVOT, while somewhat similar in function, is a separate feature. Note that Oracle's published material, including documents published in connection SQL certification exams, have described this multitable INSERT style as a pivot and makes a distinction between this pivot approach and the PIVOT keyword.**

#### CERTIFICATION OBJECTIVE 13.02

##### Merge Rows into a Table

The MERGE statement is a SQL DML statement that can combine the functionality of INSERT, UPDATE, and DELETE, all into a single SQL statement. There isn't anything you can do with MERGE that you cannot already do with some combination of those three DML statements. However, if it's possible to use MERGE as an alternative to executing two or more DML statements, then MERGE is preferable since it combines multiple DML actions into a single SQL statement, resulting in a single pass through the database. In other words, it will perform more efficiently. In many situations, such as applications built with distributed architectures, MERGE may be an invaluable weapon in your arsenal to build optimally efficient professional applications.

The syntax of MERGE follows:



```
01 MERGE INTO 'table'
02 USING table | subquery
03 ON condition
04 WHEN MATCHED THEN UPDATE SET col = expression | DEFAULT
05 where_clause
06 DELETE where_clause
07 WHEN NOT MATCHED THEN INSERT (col, col2)
08 VALUES (expr1, expr2 | DEFAULT)
09 where_clause
10 WHERE condition;
```

Note the following:

■ **Line 1** INTO specifies the target into which you are either inserting or updating rows; it can be a table or an updatable view. This line is required.

■ **Line 2** USING identifies the source of the data, which can be a table, view, or subquery. This line is required.

■ **Line 3** The ON condition for MERGE behaves essentially like a WHERE clause. It determines how to compare each row in the USING data source with each row in the MERGE INTO data target. The ON condition can use Boolean operators and expressions to form complex comparisons. In practice, the ON condition is often limited to comparing primary key values, but this is not required. This line is required.

■ **Lines 4 through 6** These lines are considered the “update clause” and identify the logic by which the MERGE will update target rows; it cannot update a column in the ON condition.

■ **Lines 7 through 9** These lines are considered the “insert clause” and identify the logic by which the MERGE will insert rows into the target table.

As you can see, it’s an involved statement. Let’s look at an example in action. Let’s say you are responsible for an application that uses the WWA\_INVOICES table (see Figure 13-7) and you have been tasked to bring in data from an outside table, ONTARIO\_ORDERS (see Figure 13-8).

FIGURE 13-7 Diagram of the WWA\_INVOICES table

WWA_INVOICES	
P *	INV_ID NUMBER (7)
	CUST_PO VARCHAR2 (10 BYTE)
	INV_DATE DATE
	NOTES VARCHAR2 (200 BYTE)
IX_SYS_C0013014	
SYS_C0013014	

FIGURE 13-8 Diagram of the ONTARIO\_ORDERS table

ONTARIO_ORDERS	
P *	ORDER_NUM NUMBER (11)
	PO_NUM VARCHAR2 (20 BYTE)
	SALES_REP VARCHAR2 (20 BYTE)
IX_SYS_C0013015	
SYS_C0013015	

The data listing for the WWA\_INVOICES table follows:

INV_ID	CUST_PO	INV_DATE	NOTES
10	WWA-200	17-DEC-19	
20	WWA-001	23-DEC-19	

The data listing for the ONTARIO\_ORDERS table follows:

ORDER_NUM	PO_NUM	SALES_REP
882	WWA-001	C. Nelson
883	WWA-017	J. Metelsky
884	NBC-201	D. Knight

The data listing for the ONTARIO\_ORDERS table follows:

Let’s use MERGE to bring the data from ONTARIO\_ORDERS into the WWA\_INVOICES table.

```
01 MERGE INTO WWA_INVOICES WWA
02 USING ONTARIO_ORDERS ONT
03 ON (WWA.CUST_PO = ONT.PO_NUM)
04 WHEN MATCHED THEN UPDATE SET
05 WWA.NOTES = ONT.SALES_REP
06 WHEN NOT MATCHED THEN INSERT
07 (WWA.INV_ID, WWA.CUST_PO, WWA.INV_DATE, WWA.NOTES)
08 VALUES
09 (SEQ_INV_ID.NEXTVAL,
10 ONT.PO_NUM, SYSDATE, ONT.SALES_REP)
11 WHERE SUBSTR(ONT.PO_NUM,1,3) <> 'NBC';
```

The preceding MERGE statement includes the following features:

■ **Line 1** Here we specify that we are going to merge rows into the table WWA\_INVOICES. Also, we assign a table alias WWA to the table WWA\_INVOICES.

■ **Line 2** Here we specify the ONTARIO\_ORDERS table as the data source and give that table an alias of ONT.

■ **Line 3** Here we define the ON condition, indicating that the columns CUST\_PO and PO\_NUM are where the common information exists that will “join” the rows logically in order to associate them with each other for the merge.

■ **Lines 4 through 5** These lines are the “update clause.”

■ **Lines 6 through 10** These lines are the “insert clause.”



- ☐ Both WHEN and ELSE are associated with their own unique INSERT statement directives; depending on which conditions apply, the appropriate INSERT statement directives will execute.
- ☐ Each condition can INSERT data in different ways into different tables.
- ☐ The INSERT FIRST statement tests each WHEN condition and executes the associated INSERT statement directives with the first WHEN condition that evaluates to true.
- ☐ The INSERT ALL statement executes all the WHEN conditions that evaluate to true.
- ☐ The ELSE clause executes for either the INSERT FIRST or INSERT ALL statement when none of the WHEN conditions has executed.
- ☐ The subquery of a multitable INSERT determines the data that will be considered in the insert logic; it can be a complex query and can include joins, GROUP BY clauses, set operators, and other complex logic.

Merge Rows in a Table

- ☐ The MERGE statement is one of the SQL DML statements, alongside SELECT, INSERT, UPDATE, and DELETE.
- ☐ MERGE replicates some of the functionality found in INSERT, UPDATE, and DELETE and combines it all into a single statement that executes with a single pass through the database.
- ☐ MERGE doesn't do anything new that you cannot already do with existing DML statements, but it does them more efficiently in combination.
- ☐ The MERGE statement includes an "update clause" and an "insert clause."
- ☐ The WHEN MATCHED THEN UPDATE keywords form the "update clause."
- ☐ The WHEN NOT MATCHED THEN INSERT keywords form the "insert clause."
- ☐ The DELETE clause of the MERGE statement only deletes rows that were first updated with the "update clause" and remain after a successful update; they must also meet the WHERE condition of the "delete clause."

SELF TEST

The following questions will help you measure your understanding of the material presented in this chapter. Choose one correct answer for each question unless otherwise directed.


Describe the Features of Multitable INSERTs

- 1 . What can an INSERT statement do? (Choose two.)
  - A. Add rows into more than one table
  - B. Add data into more than one column in a table
  - C. Delete rows by overwriting them
  - D. Join tables together
- 2 . A multitable INSERT statement:
  - A. Can accomplish tasks that cannot otherwise be done in any combination of SQL statements
  - B. Will create any tables in which it attempts to INSERT but that do not yet exist
  - C. Can use conditional logic
  - D. Is capable of inserting rows into nonupdatable views

3 . Review the following diagrams of the SPARES table:

SPARES	
SPARE_ID	NUMBER (8)
PART_NO	VARCHAR2 (30 BYTE)
PART_NAME	VARCHAR2 (80 BYTE)
IX_01	

Also examine the diagrams of the tables PORT\_INVENTORY, STORE\_INVENTORY, and SHIP\_INVENTORY, shown here.

PORT_INVENTORY	
P * NUM	NUMBER
AISLE	VARCHAR2 (7 BYTE)
PRODUCT	VARCHAR2 (15 BYTE)
LAST_ORDER	DATE
 PK_PORT_INV_NUM	

```

01 INSERT INTO SPARES
02     WHEN (PART_NO < 500) THEN
03         INTO STORE_INVENTORY (NUM, PRODUCT)
04         VALUES (SPARE_ID, PART_NAME)
05     INTO PORT_INVENTORY (NUM, PRODUCT)
06     VALUES (SPARE_ID, PART_NAME)
07     WHEN (PART_NO >= 500) THEN
08         INTO SHIP_INVENTORY (NUM, PRODUCT)
09         VALUES (SPARE_ID, PART_NAME)
10     SELECT SPARE_ID, PART_NO, PART_NAME
11     FROM SPARES;

```

Which of the following statements is true for this SQL statement?

- A. If the first WHEN condition in line 2 is true, the INTO clause in line 3 and line 4 will be executed, after which processing will skip to the next row returned by the subquery.
- B. If the first WHEN condition in line 2 is true, the WHEN condition in line 7 will not be evaluated.
- C. No matter which WHEN condition is true, the INTO clause in line 5 will be executed regardless.
- D. Regardless of whether the first WHEN condition is true, the second WHEN condition will be evaluated.

Merge Rows in a Table

7 . The MERGE statement includes a USING clause. Which of the following statements is *not* true of the USING clause?

- A. It can be used to specify a subquery.
- B. The data it identifies remains unchanged after the MERGE statement executes.
- C. The USING clause is optional.
- D. It can be used to specify an inline view.

8 . See the diagrams in question 3. You want to merge rows from the PORT\_INVENTORY table into the SHIP\_INVENTORY table. You start with the following SQL statement:

```

01 MERGE INTO SHIP_INVENTORY A
02 USING PORT_INVENTORY B
03 ON (A.NUM = B.NUM)
04 WHEN NOT MATCHED THEN INSERT
05     (A.NUM, A.AISLE, A.PRODUCT, A.LAST_ORDER)
06     VALUES
07     (B.NUM, B.AISLE, B.PRODUCT, B.LAST_ORDER)
08 WHERE TO_CHAR(A.LAST_ORDER, 'RRRR') = '2019';

```

What will this SQL statement do?

- A. It will fail with a syntax error because you must have an ELSE clause.
- B. It will fail with a syntax error because you cannot reference the target table (SHIP\_INVENTORY) in the WHERE clause in line 8.
- C. It will add rows from PORT\_INVENTORY to SHIP\_INVENTORY that do not already exist in SHIP\_INVENTORY, limited to LAST\_ORDER values from the year 2019.
- D. It will add rows from PORT\_INVENTORY to SHIP\_INVENTORY that do not already exist in SHIP\_INVENTORY, regardless of the value for LAST\_ORDER.

9 . Examine the SQL syntax in question 8. Which of the following two alternatives for line 3 are syntactically correct?

```

OPTION 1:  ON (A.NUM = B.NUM AND A.AISLE = B.AISLE)
OPTION 2:  ON (A.LAST_ORDER < B.LAST_ORDER)

```

- A. Only option 1
- B. Only option 2
- C. Both option 1 and option 2
- D. Neither option 1 nor option 2

10. Which of the following statements is false?

- A. It is possible to merge into two or more tables.
- B. It is possible to merge into a view.
- C. The USING clause can reference two or more tables.
- D. You cannot perform an update to a column that is referenced in the ON clause.

## SELF TEST ANSWERS

Describe the Features of Multitable INSERTs

- 1 . ☒ A and B. INSERT statements can add rows to more than one table using conditional and unconditional logic. INSERT statements can also add data to more than one column in any given table.
- ☒ C and D are incorrect. INSERT cannot overwrite data; it adds new

- 2

☒

C. Multitable INSERT statements can use conditional logic, with statements such as WHEN and ELSE.

☒

A, B, and D are incorrect. Multitable INSERTS do not do anything you couldn't otherwise do with one or more SQL statements. Their advantage is that they can accomplish complex SQL tasks in a single pass that might otherwise require multiple passes through the database, thus yielding performance advantages. And nothing can add rows into a nonupdatable view—if it's not updatable, it's not updatable.
- 3

☒

C. The WHEN condition in line 2 determines whether the INTO clauses in lines 3, 4, 5, and 6 will execute.

☒

A, B, and D are incorrect. The ELSE clause is not required. No particular WHEN condition is required. The INTO clause for SHIP\_INVENTORY is subject to the WHEN condition in line 2.
- 4

☒

C. The entire statement fails, and all inserted rows are rolled back. It is as if the statement had never been executed. Had this statement included any calls to a sequence generator and its NEXTVAL pseudocolumn would have advanced the count in the generator, that effect would remain unchanged. However, this example does not include any sequence generators, so that particular exception does not apply.

☒

A, B, and D are incorrect.
- 5

☒

C. The PART\_NO of 170 has a length of 3, and that is longer than 2, so the WHERE clause in line 13 is found to be true, and the row will be evaluated by the rest of the INSERT FIRST statement. Next, the PART\_NAME of PAN-OPS will cause the first WHEN condition to be true, and since this is an INSERT FIRST statement, no other WHEN condition will be considered.

☒

A, B, and D are incorrect. These answers result from various interpretations of the WHEN conditions and ELSE. In an INSERT FIRST statement, the first WHEN condition that evaluates to true is the only condition that is executed. All others are ignored. If no WHEN is found to be true, then the optional ELSE clause will be processed.
- 6

☒

D. Both WHEN conditions will be evaluated because the conditional INSERT is an INSERT ALL statement.

☒

A, B, and C are incorrect. If the first WHEN condition is true, both INTO clauses that follow it will be executed—that includes the INTO on line 5 through line 6. Whether the first WHEN condition is true or false, the second will also be evaluated since this is an INSERT ALL statement. The INTO in line 5 through line 6 will be evaluated only if the first WHEN condition is true.

Merge Rows in a Table

- 7

☒

C. The USING clause is not optional; it is required in the MERGE statement.

☒

A, B, and D are incorrect. USING can identify a table, view, or subquery. An inline view is also acceptable. It identifies the source of data to be merged; the source data remains unchanged after the MERGE statement is executed.
- 8

☒

B. It will fail because the WHERE clause references something that is not in the source table. The WHERE clause is an extension of USING, which specifies the target table. The A table alias reference is meaningless and will fail.

☒

A, C, and D are incorrect. There is no ELSE clause in MERGE, so it is not only not required, it is not accepted.
- 9

☒

C. Both options are acceptable. The ON condition can be any comparison of expressions, and it can include Boolean operators.

☒

A, B, and D are incorrect.
- 10

☒

A. You cannot MERGE into two or more tables. Only one is permitted.

☒

B, C, and D are incorrect. You can MERGE into a view provided the view is updateable. The USING clause can reference two or more tables by way of a join or subquery. You cannot change the values of a join criteria during the join, so you cannot update columns that are referenced in an ON clause.