Manipulation Statements

Introduction

In the previous learning units you learned how to affect the structure of the database, through the use of the CREATE TABLE and ALTER TABLE commands, also referred to as data definition language (DDL). This unit concentrates on affecting the data stored in the tables, using the INSERT, UPDATE and DELETE commands, which are called data manipulation language (DML). We will also discuss how to make these data changes permanent, how to undo changes, and how to lock records.

Adding Rows to a Table

To add a single row to a table using SQL, the INSERT statement is used. Only one row can be added with a single INSERT statement. The basic format is INSERT INTO tablename VALUES (value1, value2,….); If a row is to be inserted containing data for only certain columns, those columns can be referenced after the INTO keyword, then just values for those columns need to be coded in the VALUES clause. Keep in mind that the values must be coded in the same order as the columns on the table, and must be of the same data type.

To add a new book (row) to the book table, we would code:
INSERT INTO Books
VALUES (‘1439041288’, ‘ORACLE 11G SQL’, ’11-AUG-10’, 2, 45.00, 91.50, NULL, ‘COMPUTER’);

Column constraints must be adhered to when inserting rows. Primary key, foreign key, unique, check and not null constraints can possibly be violated by the statement and the DBMS will reject it. Notice this book had no discount, so we used the term NULL in the discount column value. If a column is not allowed to be null, then the DBMS will reject your insert if you try to use a null value in that column. Also notice the use of text values and numeric values, which match the column data types.

Using this same example, let’s decide that only the ISBN, title and cost are known at the time the book is to be added to the table. We are to update the book later with the other details. Our statement would change to:
INSERT INTO Books (ISBN, Title, Cost)
VALUES (‘1439041288’, ‘ORACLE 11G SQL’, 45.00);

Keep in mind that order and data type are still very important. If there is column that cannot be null (NOT NULL), then a value must be coded for that column, or the DBMS will reject the insert. Columns can be coded in any order, but the data values must be coded in the same sequence. Also, if a single quote is part of the text string, you need to code two single quotes.

To populate an existing table with rows from another table (skimming from one and inserting into another), we can use a subquery in place of the VALUES clause. Recall that a subquery is a SELECT statement that returns a result set (one or more columns and one or more rows) back to the outer query. In this case, whatever rows are returned get inserted one row at a time into the table coded on the INSERT INTO.

To illustrate, if we had a table that we wanted to contain books with a retail price over $50.00, we could "skim" the books from the Books table and insert them into our new table. Assuming the table BooksOver50 has been created already and has the same structure as the Books table, we could code:

```
INSERT INTO BooksOver50
SELECT * FROM Books
WHERE Retail > 50.00;
```

Changing Rows on a Table

The UPDATE statement is used to change the value in a column(s) for a given row or rows. Its basic format is UPDATE tablename SET columnname = value WHERE condition. Generally, it is used to change just one or a few columns at a time, based on some condition; for instance, changing the cost of all fitness books to reflect a 5% increase. If the optional WHERE clause is omitted, ALL rows are affected by the change, so be aware.

To apply the increase mentioned, we would code:

```
UPDATE Books
SET Cost = Cost * 1.05
WHERE Category = 'FITNESS';
```

As with the INSERT statement, the column and its new value must be the same data type. The value being assigned can be a constant, expression or another column. Also, violating constraints works much the same way as the INSERT statement.

More than one column can be modified in a single UPDATE statement. Back to our example, if we wanted the 5% increase to apply to both the cost and retail columns, we would change it to:

```
UPDATE Books
SET Cost = Cost * 1.05,
    Retail = Retail * 1.05
WHERE Category = 'FITNESS';
```

If management decides that ALL books are to receive the increase, we just remove the WHERE clause:

```
UPDATE Books
SET Cost = Cost * 1.05,
    Retail = Retail * 1.05;
```

Oracle allows the use of **substitution variables**, which are values that are filled in "on the fly" when the UPDATE command gets executed. This could be handy if several updates need to be done in a similar fashion. Borrowing from the previous example, if we wanted to apply a different increase percentage for each publisher id, we could code:

```
UPDATE Books
SET Cost = Cost * &increase,
   Retail = Retail * :increase WHERE
Category = ':cat';
```

Once the command is executed, Oracle interrupts and prompts for values to replace the substitution variables, making it more interactive. This technique would most likely be used if the statement is being executed from the Oracle interface, versus being imbedded in an application program.

NOTE: Oracle typically uses the ampersand (&) as the substitution operator, but Application Express uses the colon (:) as the operator.

Removing Rows from a Table

The DELETE command is used to remove one or more rows from a table. It is a simple command, but is very powerful. As with any manipulation statement, caution should be exercised, but especially with this one.

Its format is DELETE FROM tablename WHERE condition; and the WHERE clause is optional. If omitted, all rows will be deleted from the table, essentially emptying it. However, the space the rows occupied is still allocated. This is not the same as dropping a table, which completely deletes the structure and all the data from the database.

Generally, the WHERE clause is used to remove just certain rows from the table. Assume management has decided to delete all orders for some reason. To accomplish this, code the statement: DELETE FROM Orders;

To remove orders for only customer 1016, we would code:

```
DELETE FROM Orders
WHERE Customer# = 1016;
```

A point to remember about deleting records is what to do with records that are related to the record or records we are deleting. In our example, attention needs to be given to the rows in the OrderItems table that are for orders of customer 1016. The DBMS will use its constraint feature to prohibit deletion of rows in the order table if there are related rows in the OrderItems table. In this example, we would want to remove the OrderItems rows first, and then remove the Orders rows.

Transaction Control

Changes to the database using the INSERT, UPDATE and DELETE statements are not permanent until some action takes place to make them permanent. Ending the SQL Developer after executing one of the DML commands causes the DBMS to automatically save the changes to the database (assuming the Autocommit option is turned on, which is the default setting).

As an Oracle user, you have complete control over when changes to your data are to be made permanent, or are to be undone, through the use of transaction control. A transaction is considered a logical grouping of DML statements that will done together. For instance, adding a customer and their order to the JustLee Books database would require a transaction to insert rows into the Customer, Orders and OrderItems tables. After all three inserts are executed, the changes to the database should be saved.

Technically, a transaction is actually considered all commands that have been executed, but not yet committed or rolled back. Transactions formed through the use of DDL statements are automatically committed each time they are successfully executed.

To make changes permanent, we can use the COMMIT command, and to undo changes we use the ROLLBACK command. The COMMIT command is often found at the end of a script file that contains DML statements, so that after the changes are made, the commit is issued right away. The ROLLBACK command is sometimes used to reverse changes that have occurred by "accident," or changes that were made as part of a "what if" scenario. Also consider some websites that you have interacted with in the past. If you have set up a new account or changed an existing account, chances are you were given the option of saving or cancelling. These options may well have been tied to COMMIT and ROLLBACK commands.

The SAVEPOINT command is a feature that creates a virtual "bookmark" in the transaction, so that changes can be rolled back to the save point location. This may done in business settings where a transaction may get fairly lengthy and rolling back all commands in the transaction would cause considerable loss of data entry, and be cumbersome to the user.

NOTE: in order to do transaction management properly in Application Express, be sure to uncheck the AutoCommit box in the SQL window. Otherwise, every change is permanent as it is submitted.

Table Locks

Sometimes, or often times depending on the system, more than one user may need the same record at the same time. This is not a problem when the records are being browsed (retrieved with the SELECT statement). However, if the same record is being updated at the same time by different users, the DBMS must handle this situation by allowing one user to "lock" the record, and the other user to essentially "wait."

Oracle by default locks only the row that is being affected, not the entire table. It is called a shared lock, which enables other users to still "view" the record (SELECT), but not change the table structure or data for that given row. Oracle does allow whole tables to be locked, but it can be tricky and result in "deadlocks," which causes Oracle to issue error messages and to respond by releasing the lock.

Oracle supports a feature called SELECT FOR UPDATE which allows a record to be selected and held with a shared lock, anticipating that the record will also be updated by the user. This is a nice alternative to doing a normal SELECT statement, then later trying to issue an UPDATE statement only to find that another user has "locked" the record in the meantime.
