# Week 6: Subqueries

**Subqueries**

A *subquery* is a query within a query. In other words we can say that, it is a SELECT statement which is embedded in a clause of another SELECT statement. A subquery can have multiple parts in that situation the subquery answers the one part of the question whereas the parent query answer the other part.

When we have many Subqueries nested together, the inner most query is executed first and the output of that subquery is used by the query before it (outer query). A subquery can be placed in a number of SQL clauses like:

* The WHERE clause
* The HAVING clause
* The FROM clause

The syntax for the subquery is given below:

|  |
| --- |
| **Syntax:**    SELECT *select\_list*  FROM *table*  WHERE *expr operator*  (SELECT *select\_list*  FROM *table*); |

In the syntax:

***Operators:*** includes all sets of operators we have seen until now.

The example 45 below demonstrates how we can use a simple SELECT statement to create a subquery for it. Here in this example we want to display the details of books written by J.K. Rowling as we all know the name of the author or the author Id is stored in the Author table and Books table only contains information about the books. But we have a Written\_By table which stores the book id as well as the IDs of author who wrote them. Using these three table and with the help of subquery mechanism we will display the book details. Let’s see the example:

|  |  |  |
| --- | --- | --- |
| **Example 6.1: Displaying Records by using SUBQERY mechanism**   |  | | --- | | **SELECT \***  **FROM Books**  **WHERE B\_Code IN**  **(SELECT B\_Code**  **FROM Written\_By**  **WHERE Author\_id =**  **(SELECT Author\_id**  **FROM Author**  **WHERE Lname = ‘Rowling’)) ;** | |  | |
|  |

As we can see our example has two sub queries, the outermost query which gives the final result is known as the Parent query and other two are the child queries. The inner most query which fetches the author id is executed first. The output from that query is taken by the outer query and the output from the outer subquery is used by the parent query to display the list of books. This query returns multiple result set, therefore we have used IN operator to accommodate all the information. We need to be very careful about the operator we choose because sometime the final output can be multiple row as in this or it can be single row. But in case the inner query doesn’t return a result value or returns a NULL due to any condition then the outer query will consider it as NULL and no record will be displayed.

There few guidelines which must be followed when writing a subquery, these are as follows:

1. Always enclose a subquery in parenthesis.
2. To improve readability always places the subquery on the right side of the comparison condition.
3. We can use ORDER BY clause in our Subqueries for the TOP-N analysis.

**Types of Subqueries**

Oracle have categorised the Subqueries into two types, these are:

* Single-Row Subqueries
* Multiple-Row Subqueries

There is also multiple column Subqueries, which returns more than one column from the inner SELECT statement. But for now we will concentrate only on the two types mentioned above let’s have a look at them in detail.

**Single-Row Subqueries:** A single-row subquery is the one in which the inner SELECT statement returns only one row as a result. These types of Subqueries uses single-row operators with simple comparison operators ***(see Week 3 for Comparison operators)*** .the diagram below will explain clearly the concept of single-row subquery.

**Main Query**

**Subquery**

**Returns**

**Single Row**

|  |  |  |
| --- | --- | --- |
| **Example 6.2: Displaying Records by using Single-Row SUBQERY mechanism**   |  | | --- | | **SELECT Title, P\_Code, Type, Price**  **FROM Books**  **WHERE Type =**  **(SELECT Type**  **FROM Books**  **WHERE P\_Code = ‘SS’**  **AND Price >=**  **(SELECT Price**  **FROM Books**  **WHERE P\_Code = ‘TB’)) ;** | |  | |
|  |

In example 6.2 above displays the title of the books with publisher code type and price whose TYPE is same as the type of the books published by publisher SS and the price of which is greater than or equal to the price of books published by publisher TB. The outputs provided by both queries are ART and 7.25 respectively. The parent query now uses these outputs to execute itself and displays the output.

Point to be noted that both the Subqueries gives single row output therefore it is a single-row query, but if any of them would have returned multiple row output then Oracle would have thrown an exception.

|  |  |  |
| --- | --- | --- |
| **Example 6.3: Using Group function in a subquery**   |  | | --- | | **SELECT \***  **FROM Books**  **WHERE Price =**  **(SELECT MIN(Price)**  **FROM Books);** | |  | |
|  |

Example 6.3 demonstrates how we can use a group function in a query, here in this example we want to display the records of those books whose price is equal to the minimum price. As the group function after evaluation returns single row therefore it can be used with the Single-row Subqueries.

***Multiple-Row Subqueries*:** A multiple-row subquery returns more than one row of results from the subquery. For this kind we use multiple-row operators ***(see Week 3 for Comparison operators)*** in the Subqueries. In the previous type if we have provided the P\_Code value as VB the query would have failed because it returns multiple rows. Let’s take the example 46 here but this time in the second subquery just change the value of P\_Code from TB to VB

|  |  |  |
| --- | --- | --- |
| **Example 6.4: Displaying Records by using Multiple-Row SUBQERY mechanism**   |  | | --- | | **SELECT Title, P\_Code, Type, Price**  **FROM Books**  **WHERE Type =**  **(SELECT Type**  **FROM Books**  **WHERE P\_Code = ‘SS’**  **AND Price > ANY**  **(SELECT Price**  **FROM Books**  **WHERE P\_Code = ‘VB’)) ;** | |  | |
|  |

Have you noticed we got the similar result here also but the only difference is when the second subquery gets executed it returns multiple rows of output (to check it try running that block of query separately). With the help of multiple-row operator Oracle is able to provide the multiple row input to parent query for evaluation. In a very crude way after the subquery gets executed the parent query takes following form for further evaluation:

*SELECT Title, P\_Code, Type, Price*

*FROM Books*

*WHERE Type = (‘ART’) AND*

*Price > ANY (8, 11);*

The diagram below will make the concept clearer.

Returns

Subquery

Main Query

Multiple Row 1

Multiple Row *n*

Multiple Row 2

The next example 6.5 shows the use of ALL operators, like this we can use IN, ALL, ANY operator with multiple-row Subqueries. We can also use NOT operator with these multiple-row operators.

|  |  |  |
| --- | --- | --- |
| **Example 6.5: Using ALL operator with Multiple-Row Subqueries**   |  | | --- | | **SELECT Title, Price, P\_Code**  **FROM Books**  **WHERE Price < ALL**  **(SELECT Price**  **FROM Books**  **WHERE P\_Code = ‘JP’);** | |  | |
|  |

**HAVING clause with Subqueries**

It is not always necessary to use WHERE clause in Subqueries, we can also use HAVING clause as well. In this case the Oracle query executes the subquery first and returns the output to the HAVING clause of the parent query for further evaluation.

Example 6.5 demonstrates the use of HAVING clause with Subqueries, here we are taking the similar statement as we have in example 6.3 but we have made a little modification in them. Let’s have a look:

|  |  |  |
| --- | --- | --- |
| **Example 6.5: Using HAVING clause in a subquery**   |  | | --- | | **SELECT Type, P\_Code, Title, MAX(Price)**  **FROM Books**  **GROUP BY Type, P\_Code, Title**  **HAVING MAX(Price) >**  **(SELECT MAX(Price)**  **FROM Books**  **WHERE Type = ‘ART’);** | |  | |
|  |

Here in this example all the Book types which have maximum price greater than the maximum price of book type ART are displayed.

## SOLVED PRACTICE QUESTIONS

**Practice Set – 6.1**

1. **Display names of all the authors for the books available in books table.**

**Solution:**

**SELECT A.Fname, A.Lname**

**FROM Author A, Books B, Written\_By W**

**WHERE A.Author\_ID = W.Author\_ID**

**AND B.B\_Code = W.B\_Code;**

**Output:**

|  |
| --- |
|  |

1. **Display the Book name for all the books more than 2 in stock.**

**Solution:**

**SELECT B.Title**

**FROM Books B, Inventory I**

**WHERE B.B\_Code = I.B\_Code AND**

**Quantity > 2;**

**Output:**

|  |
| --- |
|  |

1. **Display the author Last names of who has written any book of type ART.**

**Solution:**

**SELECT A.Lname**

**FROM Author A, Books B, Written\_By W**

**WHERE A.Author\_ID = W.Author\_ID AND**

**B.B\_Code = W.B\_Code**

**AND B.Type =’ART’;**

**Output:**

|  |
| --- |
|  |

1. **Display the book code publisher name for all the books whose type is HOR.**

**Solution:**

**SELECT B.B\_Code, P.P\_Name**

**FROM Books B, Publisher P**

**WHERE P.P\_Code = B.P\_Code AND**

**B.Type = ‘HOR’;**

**Output:**

|  |
| --- |
|  |

1. **Display the first names of all the authors whose books are published by ST**

**Solution:**

**SELECT A.Fname, A.Lname**

**FROM Author A, Books B, Written\_By W**

**WHERE A.Author\_ID = W.Author\_ID AND**

**B.B\_Code = W.B\_Code AND**

**B.P\_Code = ‘ST’;**

**Output:**

|  |
| --- |
|  |

1. **Display Book name, publisher name and city where the price of book is more than 20.**

**Solution:**

**SELECT B.Title, P.P\_Name, P.City**

**FROM Books B, Publisher P**

**WHERE B.P\_Code = P.P\_Code AND**

**B.Price > 20;**

**Output:**

|  |
| --- |
|  |

## UNSOLVED PRACTICE QUESTIONS

**Note: For the unsolved practice question PRODUCT database is used which available in Annexure A.**

**Practice Set – 6.2**

1. Display the Product name of any department which is similar to the product delivered by S5.
2. Create a query to display Supplier ID and Suppliers name who deliver BELTs
3. Display the department number, department name of all the products whose department is on Floor 5.
4. Display the product name and price per unit of the product delivered by S1.
5. Display the delivery details for the department Accounting.
6. Display the price per unit of the products whose price is less than the product delivered to department Accounting.
7. Display the product whose supplier is not S5 and whose price per unit is less than the product delivered by S1.
8. Create a query to display the department name whose gets the product with second letter as ‘*U*’.

\*\*\* Chapter Ends \*\*\*

# Week 7: Producing Readable Outputs with iSQL \* Plus

**Background**

## SOLVED PRACTICE QUESTIONS

**Practice Set – 7.1**

## UNSOLVED PRACTICE QUESTIONS

**Note: For the unsolved practice question PRODUCT database is used which available in Annexure A.**

**Practice Set – 7.2**

\*\*\* Chapter Ends

# Week 8: Manipulating Data

**Background**

As we have already discussed in week one that DML is a core part of SQL and provides various statements to access and modify database tables. Although SELECT statement is the most useful statement among various DML statements; but other statements are also equally important. It is not always the case that a database user needs to display information, sometimes he/she might want to insert new information into the database or delete the old ones, or even might want to modify some information; in order to perform these activities on a database table we need to know few more DML statements.

**Insert Statement**

To add a new row to an existing table or to a new table we make use of the INSERT statement. The syntax for the INSERT statement is as follows:

|  |
| --- |
| **Syntax:**  INSERT INTO tablename[(column1,column2[,column x])]  VALUES (value1,value2 [,value x]); |

In the syntax:

***INSERT INTO*** Keyword to insert data values in a table

***Tablename*** name of the table in which data is inserted

***Column1…x*** name of the columns in the table to populate

***Value1…x*** corresponding values for the column

***VALUES*** clause which adds only one row at a time in a table

All the character and date values are always enclosed within the single quotation mark and the numeric values should not be enclosed in single quotation mark as implicit data conversion may take place. Basically SQL INSERT INTO syntax has two main forms and the result of either of them is same.

The first form of INSERT INTO syntax doesn’t specify the column names where data will be inserted but just their value. In this type it is assumed that all the data entries are entered are in their default order of the column in the table and a value must be provided for each column , if any of them is missing then the Oracle will throw an exception.

|  |
| --- |
| **Syntax Type 1:**  INSERT INTO Tablename  VALUES (value1, value2, value x) |

The second form of INSERT INTO syntax specifies both the column names along with their values. The number of columns must match will the number of values otherwise Oracle will throw an exception.

|  |
| --- |
| **Syntax Type 2:**  INSERT INTO Table1 (Column1, Column2, Column x)  VALUES (Value1, Value2, Value x) |

This syntax type is similar to the actual syntax of the INSERT INTO statement.

Rows in a table can be added with specific values or rows can be created from existing data using a subquery. There are many ways of writing a simple INSERT INTO statement. Let’s discuss them one by one.

**Inserting New Rows**

This is the most typical and general type of insertion. For this kind we can use any of the two types discussed above to insert the data into the table. Example 8.1 and 8.2 shows the both type. To find out whether the statement is correct or not we can always use the SELECT statement to check the newly inserted data.

|  |  |  |
| --- | --- | --- |
| **Example 8.1: INSERT INTO statement with syntax Type 1**   |  | | --- | | **INSERT INTO Author**  **VALUES (26,'Chetan','Bhagat','New Delhi');** | | **1 row created** | |
|  |

|  |  |  |
| --- | --- | --- |
| **Example 8.2: INSERT INTO statement with syntax Type 2**   |  | | --- | | **INSERT INTO Author (Author\_id, Fname, Lname, City)**  **VALUES (27,'Ruskin','Bond','New Delhi');** | | **1 row created** | |
|  |

The above two examples are alright when we are sure about the values which needs to be inserted into the table, but what happens when there is a column which doesn’t have any value to it.

The answer to that question is to enter NULL as value to those columns. There are two methods in which we can insert a NULL value into the table, but discussing that we have to be sure of one thing that the column for which we are entering NULL value can take it; this can be checked by using the DESCRIBE command which will show us the table schema with the constraints.

**Method to insert NULLs**

* IMPLICIT METHOD: In the *implicit method* the column name which has NULL values are omitted and only those Columns are mentioned which have their respective values. Example 8.3 demonstrates the use of *implicit method*.

|  |  |  |
| --- | --- | --- |
| **Example 8.3: Inserting Row with Null Value using *IMPLICIT* method**   |  | | --- | | **INSERT INTO Author (Author\_id, Fname, Lname )**  **VALUES (27,'Ruskin','Bond');** | | **1 row created** | |
|  |

In the above example the column name CITY has been removed from the column list and also there is no corresponding value for that column, while insertion the Oracle server will insert value for Author id, first name and last name but the field City will be blank.

* EXPLICIT METHOD: In the *explicit method* we have to specify the NULL keyword in the VALUES clause of the insert statement. Example 8.4 demonstrates the use of explicit method.

|  |  |  |
| --- | --- | --- |
| **Example 8.4: Inserting Row with Null value using *EXPLICIT* method**   |  | | --- | | **INSERT INTO Author**  **VALUES (27,'Ruskin','Bond',' ');** | | **1 row created** | |
|  |

In the above example as you may have noticed we haven’t mentioned column names, we are just passing the values. As it a character value so we have to specify empty string, same is done if it’s a date value but if it is a numeric value then we have to write the keyword NULL without any single quotation mark.

**Inserting rows from User Input**

By using the substitution variable we can take input to be inserted into the table at the run time. We have seen how we can use substitution variable in previous week notes. We can use substitution with the INSERT statements also when we want user to provide value for some column.

|  |  |  |
| --- | --- | --- |
| **Example 8.5: Inserting Row with user input**   |  | | --- | | **INSERT INTO Author (Author\_id, Fname, Lname)**  **VALUES (&Author\_id, ’&Fname’, ‘&Lname');** | |  | |
|  |

In the above example when the INSERT statement with a substitution variable is executed the Oracle server prompt the user to enter the values for the fields one after another. When the final column value is inserted it shows the *old values* and *new values.* A new row is inserted in this way by asking user input.

**Inserting Rows by Copying Rows from another table**

We can use INSERT statement to insert values from another table by writing a subquery in place of the VALUES clause. The example 8.6 shows how we can use the subquery to insert values from another table.

|  |  |  |
| --- | --- | --- |
| **Example 8.6: Inserting Row by copying rows from another table**   |  | | --- | | **INSERT INTO New\_Books(B\_Code, B\_Title, Price)**  **SELECT B\_Code, Title, Price**  **FROM Books;** | | **34 rows created** | |
|  |

To make use of this type of insertion we have to make sure that the table in which data is being copied is already created otherwise Oracle will throw an exception. We can also restrict our insertion by putting a WHERE clause in the subquery.

**UPDATE Statement**

The SQL UPDATE statement is used to change the data in the already existing database row(s) and usually we need to add a SQL WHERE clause in our UPDATE statement in order to specify which row(s) we want to change. The syntax for the UPDATE statement is given below:

|  |
| --- |
| **Syntax:**    UPDATE tablename  SET column1 = value1 [, column2 = value2]  [WHERE condition]; |

In the syntax:

***Tablename*** is the table in which we want to update data

***Column*** name of the column in table to be updated

***Value*** new which is assigned to the column

***Condition*** identifies the columns to be updated

**Updating Rows in a Table**

The Update statement updates a row with new set of data; if WHERE clause is specified with the UPDATE statement then the row which satisfies the condition gets updated. If the WHERE clause is not present in the UPDATE statement then all the rows in the will be updated.

To demonstrate the use of UPDATE and DELETE statements we are creating a new table named as New\_Author which will be similar to the Author table of Books database with the similar schema and data. Example 8.7 and 8.8 shows the use of update statement with and without WHERE clause

|  |  |  |
| --- | --- | --- |
| **Example 8.7: Updating rows with WHERE clause**   |  | | --- | | **UPDATE New\_Books set Fname = ‘Mr Chetan’**  **WHERE Author\_id = 26;** | | **1 row updated** | |
|  |

|  |  |  |
| --- | --- | --- |
| **Example 8.8: Updating rows without WHERE clause**   |  | | --- | | **UPDATE New\_Books set Fname = ‘Mr Chetan’;** | | **26 rows updated** | |
|  |

As you must have noticed that in example 8.7 only one row is update but in example 8.8 all the 26 rows in the New\_Author table are updated and when we check is by using SELECT statement the First names of all the Authors have been changed to Mr Chetan. Therefor it is very useful to make use of WHERE clause when we want only some specific data in a table to be updated.

The above two examples were simple two ways in which we can update rows. There are also some other ways in which a row can be updated, even we can update two or more rows together. The example 8.9 below shows how we can use UPDATE statement with a subquery to update two or more columns and update from the rows based on another table.

|  |  |  |
| --- | --- | --- |
| **Example 8.9: Updating rows using Subqueries**   |  | | --- | | **UPDATE New\_Author set Fname = (SELECT Fname FROM**  **New\_Author WHERE Author\_id= 21)**  **WHERE Author\_id = (SELECT Author\_id FROM**  **Author WHERE Lname= ‘Bhagat’) ;** | | **1 row updated** | |
|  |

In the above example we have updated our New\_Author table using subquery and the values taking from another table. The first subquery selects the first name of author whose id is 21 to be set as newly changed value. In the second subquery we are putting a condition for the UPDATE statement which is the value taken from another table.

One of the important points which need to be taken care of is the integrity constraints i.e. whenever we update a value and if the value is bounded with a constraint then the Oracle will throw an error. For example we if we try to change or update the value of Author\_id in Written\_By table then we will encounter an exception because Author\_id is a Primary key in Author table which is used to create relation with other tables in the database, so if that value get changed then there will be redundancy of data to avoid it Oracle checks for the constraint and executes query accordingly.

**DELETE Statement**

So far we have seen how to SELECT data from the database table and how to insert and update data from the database table. What if we want to remove some data from the table? To remove an existing row from the database table we make use of DELETE statement. The syntax for DELETE statement is given below:

|  |
| --- |
| **Syntax:**    DELETE FROM tablename  [WHERE condition]; |

In the syntax:

***Tablename*** name of the table from which data is to be removed

***Condition*** to identify the rows to be deleted in particular

The WHERE clause is optional; if it is used then only those rows are deleted which satisfy the WHERE clause condition. If it is not used in DELETE statement then all the rows from the table are deleted and we are only left with empty table.

The example below demonstrates the use of DELETE statement Example 8.10 is with WHERE clause and Example 8.11 is without WHERE clause.

|  |  |  |
| --- | --- | --- |
| **Example 8.10: Deleting rows with WHERE clause**   |  | | --- | | **DELETE FROM New\_Author**  **WHERE Author\_id = 26;** | | **1 row deleted** | |
|  |

|  |  |  |
| --- | --- | --- |
| **Example 8.11: deleting rows without WHERE clause**   |  | | --- | | **DELETE FROM New\_Author;** | | **26 rows deleted** | |
|  |

In DELETE statements if we try to delete a record which is bounded with an integrity constraint we receive an exception from Oracle. For example if we try to delete a row from Books table we will encounter an error because B\_Code from Books table act as a foreign key in Written\_By and Inventory table. Therefore to delete any record from Books table we have to be sure that all the records referring the particular B\_Code in Written\_By and Inventory tables are deleted first.

**MERGE Statement**

The SQL MERGE statement is used to select rows from one or more sources for update or insertion into a table or view. We can specify condition to determine whether to insert or update a row in the target table. This statement is a convenient way of combining multiple operations but it also doesn’t allow us to perform UPDATE on the same row multiple times in the same MERGE statement. The syntax for the MERGE statement is given below:

|  |
| --- |
| Syntax:    MERGE INTO tablename table\_alias  USING (table | view | subquery) alias  ON (join condition)  WHEN MATCHED THEN  UPDATE SET  Col1 = value1,  Col2 = value2  WHEN NOT MATCHED THEN  INSERT (column-list)  VALUES (Column-values); |

In the syntax:

|  |  |
| --- | --- |
| ***INTO clause*** | specifies the target table being used for update and insertion |
| ***USING clause*** | identifies the source of the data to be inserted or updated |
| ***ON clause*** | condition based on which MERGE statement updates or inserts |
| ***WHEN MATCHED |***  ***WHEN NOT MATCHED*** | Instructs the server how to respond to the result of the join condition |

To demonstrate the use of MERGE statement we are taking into consideration the New\_Books table. Let’s suppose that the table has been created and there is no record in the table so the example 8.12 shows us how to use MERGE statement to insert and update a table.

|  |  |  |
| --- | --- | --- |
| **Example 8.12: Using MERGE statement**   |  | | --- | | **MERGE INTO New\_Books NB**  **USING Books B**  **ON (NB.B\_Code = B.B\_Code)**  **WHEN MATCHED THEN**  **UPDATE SET**  **NB.B\_Title = B.Title,**  **NB.Price = B.Price**  **WHEN NOT MATCHED THEN**  **INSERT VALUES (B.B\_Code, B.Title, B.Price);** | | **34 rows merged** | |
|  |

**Database Transactions**

One of the most important features of database is data consistency, based on the transactions. Transaction consists of DML statement based on which the actions are taken to control a transaction.

Controlling a transaction involves coordinating multiple concurrent accesses to the same data. For example Mr A and Mr B have account in a bank, at some point in time Mr A decides to transfer some X amount into Mr B’s account. This transferring of money involves two steps:

1. Debit Mr A’s account by X amount
2. Credit Mr B’s account by X account

Both of these step has to happen together in order to maintain the consistency of the data. Both the actions should either fail or succeed. The credit should not happen without debit. To avoid such problem another set of statement are introduced which are specifically used to control the data known as ***Transaction Control Language***.

Basically a transaction starts when DDL or DML statement is issued and ends when one of the following occurs:

* A DDL statement like CREATE is issued
* A TCL statement is issued
* The user exists the iSQL\* Plus
* A machine fails/ system crashes

When a transaction ends, the new SQL statement automatically starts a new transaction. A TCL consist of various data/transaction control statements these are as follows:

1. **COMMIT**: This statement ends the transaction by making all the pending data changes permanent. Once this statement is issued all the data which was there in buffer is removed and stored in memory.

We can commit a transaction by issuing ***COMMIT*** statement after any of the SQL statement.

1. **SAVEPOINT**: This statement is used to define breakpoints for a transaction to allow partial rollbacks. It divides the transaction into smaller parts. Multiple savepoints can exist in a single transaction.

A savepoint can be issued by using ***SAVEPOINT <name>*** statement.

1. **ROLLBACK**: This statement ends the transaction by discarding all the pending data changes available in the Buffer.

We can discard a transaction by issuing ***ROLLBACK*** statement after any of the SQL statement.

1. **ROLLBACK TO SAVEPOINT**: This statement rolls back the transaction to the specified savepoint, it discard any changes made after the specified savepoint; even it discard any savepoint created after this particular savepoint. If we choose only the Rollback option without any savepoint the entire transaction is rolled back, as savepoint are only logical

For example we made some changes in the New\_Author table using update statement. Therefore after the SQL UPDATE statement we can create a new savepoint and named it A. After that we delete a record and after deleting we realise that we need that record back in this type of situation we can use the statement ROLLBACK TO A.

So all the deleted information is revert back to the table, as we have performed only one action after declaring the savepoint therefore only that is discarded if we would have issued more SQL statements, then those statements would also be reverted till the point where we have declared our savepoint A. But if we have issued a simple ROLLBACK command then everything including the update statement would have been discarded. This structure of the statement looks like this

|  |
| --- |
| Update ….  Rows updated  Savepoint A  Savepoint created  Delete …  Row deleted  ROLLBACK TO A  Rollback complete |

The above mentioned statements help us to explicitly control any transaction; but we can also implicitly control transaction processing by using the ***AUTOCOMMIT*** statement which be set **ON** or **OFF**, if set to ON then after every DML statement is executed it commits the transaction and then we cannot rollback the changes.

Similarly there is an automatic rollback option which is automatically executed whenever a transaction is interrupted by system failure.

This week we have learned about various DML statement and the statements to control the transaction the table below show the summarised description of all the statements.

|  |  |
| --- | --- |
| **Statement** | **Description** |
| INSERT | Adds a new row to the table |
| UPDATE | Modifies existing rows in the table |
| DELETE | Removes existing rows from the table |
| MERGE | Conditionally inserts or updates data in a table |
| COMMIT | Makes all pending changes permanent |
| ROLLBACK | Discards all pending data changes |
| SAVEPOINT | Is used to roll back to the savepoint marker |

## SOLVED PRACTICE QUESTIONS

**Practice Set – 8.1**

1. **Describe the structure of the New\_Books table to identify the column names.**

**Solution:**

**DESC New\_Books;**

**Output:**

|  |
| --- |
|  |

1. **Add the first row of data to the New\_Books table from the following sample data. Do not list the columns in the INSERT statement.**

|  |  |  |
| --- | --- | --- |
| **B\_CODE** | **B\_TITLE** | **PRICE** |
| **1** | **Ghost Story** | **9.79** |
| **2** | **Death of Kings** | **9.49** |
| **3** | **Gamble** | **13.99** |
| **4** | **Master chef kitchen bible** | **20.00** |

**Solution:**

**INSERT INTO New\_Books VALUES (‘1’, ‘Ghost Story’, 9.79);**

1. **Populate the New\_Books table with the second row of sample data from the preceding list. This time, list the column explicitly in the INSERT statement.**

**Solution:**

**INSERT INTO New\_Books (B\_Code, B\_Title, Price)**

**VALUES (‘2’, ‘Death of Kings’, 9.49);**

1. **Display and confirm your addition to the table**

**Solution:**

**SELECT \* FROM New\_Books;**

**Output:**

|  |
| --- |
|  |

1. **Populate the table with the next two rows of sample data by asking the user input at the time of insertion.**

**Solution:**

**INSERT INTO New\_Books (‘B\_Code’, ’B\_Title’, Price)**

**VALUES (‘&Book\_Code’, ‘&Book\_Title’, &Book\_Price);**

1. **Confirm and display the addition to the table.**

**Solution:**

**SELECT \* FROM New\_Books;**

**Output:**

|  |
| --- |
|  |

1. **Which of the following statement is used to make data permanent, use this statement to make addition done above permanent?**
2. **SAVE**
3. **LOCK**
4. **COMMIT**
5. **ROLLBACK**

**Solution:**

**(C) COMMIT is the statement used to make data permanent into the table.**

1. **Which of the following are DML statements? (Choose all that apply)**
2. **COMMIT**
3. **MERGE**
4. **CREATE**
5. **DELETE**
6. **UPDATE**
7. **DROP**

**Solution:**

**(A, B, D, E) Because CREATE and DROP are the Data Definition Language (DDL) statements.**

1. **Change the book title of bode code 3 to ‘Mini Shopaholic’ and verify the changes.**

**Solution:**

**UPDATE New\_Books SET B\_Title = ‘Mini Shopaholic’**

**WHERE B\_Code = ‘3’;**

**Output:**

|  |
| --- |
|  |

1. **Increase the price of the books by £ 1 where price is less than £ 10.**

**Solution:**

**UPDATE New\_Books SET Price = Price + 1**

**WHERE Price < 10;**

1. **Mark an intermediate point in the processing of the transaction and name it as update\_Books**

**Solution:**

**SAVEPOINT update\_Books;**

1. **Which of the following statement is correct?**
2. **DELETE ALL FROM New\_Books;**
3. **DELETE \* FROM New\_Books;**
4. **DELETE FROM New\_Books;**

**Solution:**

**(C) Because DELETE statement does not take any argument to delete records from the table it only considers the condition to delete a specific record.**

1. **Empty the entire New\_Books table and also confirm that the table is empty.**

**Solution:**

**DELETE FROM New\_Books;**

**SELECT \* FROM New\_Books;**

**Output:**

|  |
| --- |
| **No rows selected** |

1. **Which of the following transaction control statement will be used to discard most recent change in the New\_Books?**
2. **Commit;**
3. **SAVEPOINT update\_Books;**
4. **ROLLBACK:**
5. **ROLLBACK TO update\_Books;**

**Solution:**

**(D) It will roll back the transaction to savepoint update\_Books discarding any changes after that point.**

1. **What will happen to the transaction if we issue the following statement:**

**ROLLBACK;**

1. **Discard and removes all the data in buffer permanently**
2. **Restore all the data in the buffer till the last savepoint**
3. **Save the data in buffer permanently**

**Solution:**

1. **Discard and removes all the data in the buffer permanently and all the changes made to the table are lost forever if it is not committed before using this statement.**

## UNSOLVED PRACTICE QUESTIONS

**Note: For the unsolved practice question PRODUCT database is used which available in Annexure A.**

**Practice Set – 8.2**

1. Create a temporary table named as DELIVERY\_TEMP which is based on the DELIVERY table but contains details of only TOYS department.
2. Try using the UPDATE command to change the delivery dates in DELIVERY\_TEMP table.
3. Delete all the delivery details of Supplier S5 and select from it.
4. Delete anything else remaining in the DELIVERY\_TEMP table and select from it.
5. Remove the DELIVERY\_TEMP table.

\*\*\* Chapter Ends \*\*\*