# Week 4: Displaying Data from Multiple Tables

**Background**

We very well know that a database is a collection of many tables that stores data. Till we have seen how we can display data from one table based on our requirements. But sometimes the need arises to display data from two or more different tables. In that situation we need some mechanism which can bring together the data from two separate tables.

In relational database, related data can be stored in multiple tables. It also provides ability to join two or more tables so that some relevant information can be displayed. This week we will focus on these joining techniques for querying data from more than one table. Later in week 6 we will discuss the other techniques of querying data from multiple tables.

**Joins**

A *join* is a query that combines rows from two or more tables or views (discussed in week 11). Oracle performs a join whenever multiple table appear after the FROM clause in a SELECT statement. The Diagram below shows the concept of joining two tables.



In order to join two tables we need to establish relationship between certain columns in these tables, that is usually the *PRIMARY KEY* and *FOREIGN KEY* columns. The syntax for joining table is as follows:

|  |
| --- |
| **Syntax:**    SELECT Table1.column, Table2.column  FROM Table1, Table2  WHERE Table1.column1 = Table2.column2; |

In the Syntax:

***Table1.column*** table and column from which data is retrieved

***Table1.column1 = Table2.column2*** condition that joins (or relates) the tables together

There are few guidelines which are needed to be followed when joining two tables, these are listed below:

* When writing the SELECT statement that joins tables, precede the column name with the table name for clarity and to enhance the database access.
* If the same column name appears in more than one table, the column name must be prefixed with the table name.
* The most important guideline is, to join *n* tables we need to have *n-1* join condition in the WHERE clause. For example if we want to join five tables then we need to have minimum of 5-1 condition in the WHERE clause.

If the multiple tables have same column names, the duplicate column names should be qualified in the queries with their table name or table alias.

In week one, we spoke about the column aliases, here in this week we will discuss it further with the aliases for the table. Table alias name can be specified next to the table name separated with a space. It helps in increasing the readability of the query and can also be used to shorten the long table names with shorter alias name. Although it is an easy to follow concept, but still follows some guidelines, these are as follows:

* Table alias can be up to 30 character in length, but shorter the better.
* If a table alias is used for a particular table name in the FROM clause, then that table alias must be substituted for the table name throughout the SELECT statement.
* Table alias should be meaningful.
* The table alias is only valid for the current SELECT statement.

|  |  |  |
| --- | --- | --- |
| **Example 4.1: Using Table Alias name to Join tables**   |  | | --- | | **SELECT B.Title, B.P\_Code, P.P\_Name**  **FROM Books B, Publisher P**  **WHERE P.P\_Code = B.P\_Code;** | |  | |
|  |

Example 4.1 demonstrate the use of table alias, here we have used B as an alias to Books table and P as an alias to Publisher table. To identify which column is from which table we have prefixed the table alias name to the column name like B.P\_Code, as we all know that P\_Code is also present in the Publisher table, if we don’t provide any alias then the oracle will get confused and will throw an exception. To avoid this ambiguity we always provide the table name or the table alias name in a multiple table query. One more thing to notice over here is that we are joining two tables therefore the number of condition in the WHERE is 2-1 = 1 i.e. we have got one condition in our WHERE clause, hence we are meeting all the criteria need to join tables.

**Types of Joins**

Oracle offers various join techniques but the most prominent and widely used are listed below:

* Equijoin
* Non-equijoin
* Outer join
* Self-join

The above mentioned join types follow the same guidelines and helps in displaying data from multiple tables in different ways. Let’s discuss each one of them one by one.

**Equijoin**

The most common operator used to relate two tables is the equality operator (=). If we relate two tables using the equality operator, it is then referred as the *equality join* and also known as *Equijoin.* This type of join combines rows from two tables that similar values for the specified columns.

|  |  |  |
| --- | --- | --- |
| **Example 4.2: Using EQUIJOIN to display data from multiple tables**   |  | | --- | | **SELECT I.B\_Code, B.Title, B.Price, I.Quantity**  **FROM Inventory I, Books B**  **WHERE I.B\_Code = B.B\_Code;** | |  | |
|  |

Equijoins are also known as *inner join* and *simple join,* because it return only those rows which satisfy the join condition in the WHERE clause. Frequently this type of join requires Primary key and foreign key as they establish a relationship between the two tables.

The example 4.2 above shows the use of Equijoin to display the data from two different tables. Both the table have a common column which is used to establish the relation between them i.e. the B\_Code; it’s the primary key in the Books table and act as foreign key in the Inventory table and also share the same value for a row. Here in this example in the SELECT part we have mentioned all the columns whose data we need to display following the table alias convention and in the WHERE part we are using the equality operator to establish a join condition, here in the join condition the Oracle compares the values of B\_Code in Inventory with those in Books and the rows which satisfies the condition are displayed in the output.

In addition to the join condition we can have certain other criteria for the WHERE clause to restrict the rows to be displayed. This can be done by using AND operator with the equality operator.

The example 4.3 below demonstrates the restricted display of data from multiple tables. The example is same as it was in the example 35 but with an extra AND condition.

|  |  |  |
| --- | --- | --- |
| **Example 4.3: Using EQUIJOIN with AND operator to display data from multiple tables**   |  | | --- | | **SELECT I.B\_Code, B.Title, B.Price, I.Quantity**  **FROM Inventory I, Books B**  **WHERE I.B\_Code = B.B\_Code**  **AND I.Quantity = 3;** | |  | |
|  |

The example here demonstrates the books with quantity 3, where Title and Price is taken from the Books table and Quantity is taken from the Inventory table and B\_Code is the common column in both the tables. These kinds of joins are also known as *complex joins*.

**Non-equijoin**

A non-equijoin joins is a type of join in which we can join two tables with the join condition where the join condition uses other than equal operator (=).

|  |  |  |
| --- | --- | --- |
| **Example 4.4: Using NON-EQUIJOIN to display data from multiple tables**   |  | | --- | |  | |  | |
|  |

**Outer Join**

An OUTER join is used to return all the rows that exist in one table, even though corresponding rows do not exist in the joined table. The (+) is used to denote an Outer join in a query. The (+) is placed at the end of the table name in the WHERE clause. The table with (+) should be the table that does not have matching rows. The syntax for the outer join is given below:

|  |
| --- |
| **Syntax:**    SELECT table1.column, table2.column  FROM table1, table2  WHERE table1.column[(+)] = table2.column [(+)] |

In many implementations, the Outer join is broken down into joins called:

**LEFT OUTER JOIN:** It’s a join between two tables that returns rows based on the matching conditions as well as unmatched rows from the table with no (+) at the end (i.e. table1.column(+) = table2.column).

**RIGTH OUTER JOIN:** It’s a join between the two tables that returns rows based on the matching condition as well as unmatched rows from the table with no (+) sign at the end (i.e. table1.column = table2.column(+)).

**FULL OUTER JOIN:** This is a new type of join to Oracle 9i, but this join also joins the two tables that return rows based on the matching condition as well as unmatched rows from both the tables on the right and left of the equality operator in the Join condition. This join can be achieved by using UNION operator along with the outer join operator (+).

The outer join operator (+) can be used on the one side of a join condition, however we can use OUTER join on more than one column of the same table in the join condition.

The concept of the OUTER join is explained in the example 4.5 below:

|  |  |  |
| --- | --- | --- |
| **Example 4.5: Using OUTER JOIN to display data from multiple tables**   |  | | --- | | **SELECT I.B\_Code, B.Title, B.Price, I.Quantity**  **FROM Books B, Inventory I**  **WHERE I.B\_Code (+) = B.B\_Code;** | |  | |
|  |

In the example above all the books which are available in the inventory are displayed as well as those which are not present are also displayed. If we remove the (+) from the join condition then we will only receive the data of the books which are available in the Inventory table.

**Self-Join**

A self-join is a join of a table to itself. This table appears twice in the FROM clause and is followed by the table aliases that qualify column names in the join condition.

|  |  |  |
| --- | --- | --- |
| **Example 4.6: Using SELF-JOIN to display data from multiple tables**   |  | | --- | | **SELECT first.Author\_ID, Second.Author\_ID, first.B\_Code**  **FROM Written\_By first, Written\_By Second**  **WHERE first.B\_Code = Second.B\_Code;** | |  | |
|  |

## SOLVED PRACTICE QUESTIONS

**Practice Set – 4.1**

1. **Write a query to display author’s last name, Publisher name and city for all the books.**

**Solution:**

**SELECT A.Lname, P.P\_Name, P.City**

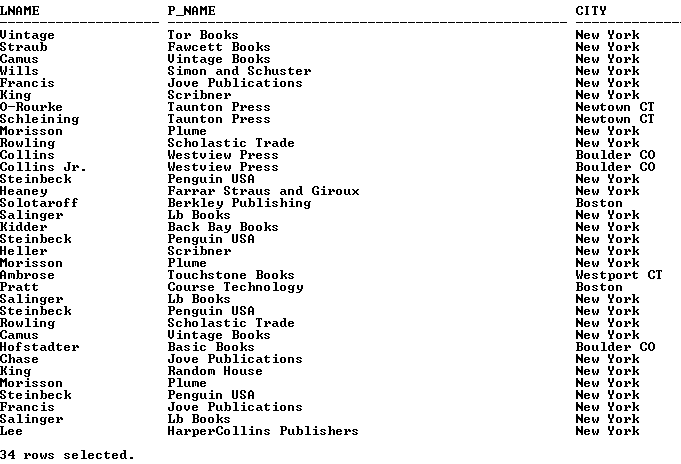
**FROM Author A, Publisher P, Books B, Written\_By W**

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

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

**P.P\_Code = B.P\_Code;**

**Output:**



1. **Crate a unique listing of the book name published by** Penguin USA. Include the publishers name in the output.

Solution:

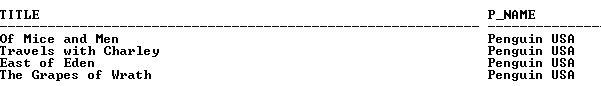
SELECT B.Title, P.P\_Name

FROM Books B, Publisher P

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

P.P\_Name = ‘Penguin USA’;

Output:



1. **Write a query to display Authors last name, Publisher name and city of all the books with type ART.**

**Solution:**

**SELECT A.Lname, P.P\_Name, P.City**

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

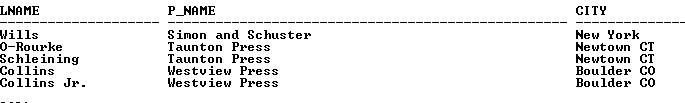
**WHERE A.Author\_ID = W.Author\_ID AND**

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

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

**B.Type =’ART’;**

**Output:**



1. **Display the name of books and their publisher name for all the books whose publisher have ‘a’ as second character in their names.**

**Solution:**

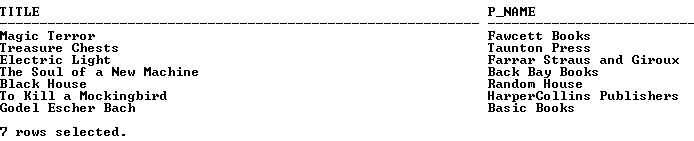
**SELECT B.Title, P.P\_Name**

**FROM Books B, Publisher P**

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

**P.P\_Name LIKE ‘\_a%’;**

**Output:**



1. **Write a query to display the First name of author, books name and the release date for all the books of type ‘MYS’**

**Solution:**

**SELECT A.Fname, B.Title, W.Release\_Date**

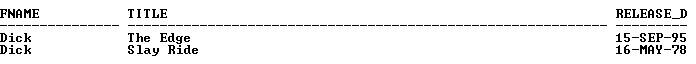
**FROM Author A, Books B, Written\_By W**

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

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

**B.Type = ‘MYS’;**

**Output:**



## UNSOLVED PRACTICE QUESTIONS

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

**Practice Set – 4.2**

1. Write a query to display the suppliers name, department number, department name for all the products.
2. Create a unique listing of all the departments’ and the floor 2. Include the supplier for the department.
3. Write a query to display the product name, department name where the quantity is more than 20 units.
4. Write a query to display product name, department name and supplier name of the product whose quantity is in the range of 10-35 units
5. Display all the product name and supplier Id for all the products who have *a* as third character.
6. Display the department number, department name, phone of all departments who receive goods from Supplier S4.
7. Display the product name along with their department number and name.
8. Create a query to display the product name, quantity, department name and supplier’s name.

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

# Week 5: Aggregating Data Using Group Functions

**Background**

We have discussed earlier in week three about the Single-row functions which take zero or more arguments and returns a single value. Also in week three we mentioned about various types of function available in Oracle and for our knowledge we decided to discuss two main types for our understanding, these are *single-row functions and Aggregate multiple row function*. We have already got a fair idea about the single-row functions. Now in this week we will focus on the other type also known as GROUP FUNCTIONS

Group functions are very different from single-row function, single-row functions are evaluated once for each row retrieved whereas group functions are evaluated on groups of one or more rows at a time. The diagram below makes the concept more understandable

|  |  |
| --- | --- |
| **B\_CODE** | **PRICE** |
| 180 | 7.25   |  | | --- | | **HIGHEST PRICE** | | **24.95** | |
| 189 | 7.95 |
| 200 | 8 |
| 378 | 24.5 |
| 079X | 24.95 |
| 808 | 6.95  The expensive book in the BOOKS table |
| 1351 | 19.6 |
| 1382 | 24.45 |
| 138X | 12.95 |
| 2226 | 13.95 |

By applying a group function to a set of values we can find out the expensive book in the database.

**Group Function Fundamentals**

Group functions are also known as ***aggregate functions***, which return a single result row based on group of rows. These functions can appear in the SELECT list and in ORDER BY and HAVING clauses.

Aggregate functions are commonly used by the GROUP BY clause in a SELECT statement. In a query containing a GROUP BY clause, the elements of the select list can be aggregate functions, GROUP BY expressions, constants, or expressions involving one of these. Oracle applies the aggregate functions to each group of rows and returns a single result row for each group.

If we omit the GROUP BY clause, then Oracle applies aggregate functions in the select list to all the rows in the queried table or view. We can use aggregate functions in the HAVING clause to eliminate groups from the output based on the results of the aggregate functions, rather than on the values of the individual rows of the queried table or view.

The syntax of the SELECT gets modified when we use group function or the group by clause. The updated syntax is given below. If you notice slowly we are reaching the complete SELECT statement syntax.

|  |
| --- |
| **Syntax:**    SELECT [column1,] group\_Function(column), . . .  FROM tablename1 [, tablename2]  [WHERE condition] [And | or | LIKE condition . . .]  [GROUP BY column-list]  [ORDER BY “condition-list” [ASC | DESC]] |

As we have said earlier that all the aggregate function takes multiple values but there are few exceptions like DISTINCT forces the aggregate function to consider only distinct values of the argument expression whereas ALL makes it consider every value including the duplicates. Default is ALL, if we don’t specify DISTINCT then it consider every value in the expression.

Hence there are few more guidelines which need to be followed when using Group/aggregate function. These are as follows:

* The data types for the functions with the expr argument may be CHAR, VARCHAR2, NUMBER or DATE.
* All group functions ignore NULL values. To substitute a value for NULL values use the NVL, NVL2 or COALESCE functions.
* The Oracle server implicitly sorts the result set in ascending order when using a GROUP BY clause. Use ORDER BY clause if the need arises to sort the data in descending order.

**Types of Group Function**

As with Single-row function, Oracle offers a wide variety of multiple row functions. These functions appear in the SELECT clause or HAVING clause (discussed later in notes) of SELECT statement. When used in the SELECT clause, they usually require a GROUP BY clause, but if GROUP BY clause is not specified the entire grouping is for the entire result set.

One very important point is to be kept in mind is that a group function cannot appear in WHERE clause. The table below shows the aggregate function with their descriptions and example.

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **Purpose** | **Example** | **Result** |
| **AVG**(*[DISTINCT | ALL] n*) | The AVG function returns the mean of numeric expression *n*. If neither DISTINCT nor ALL is specified in the function call, the default ALL is taken into consideration. | **SELECT AVG(Price) AS “Average Price” FROM Books;** |  |
| **COUNT**(*{\*| [DISTINCT | ALL] expr}*) | This function returns the number of rows in the query. The asterisk (\*) is a special quantity, it counts all rows in the result set regardless of NULLs. | **SELECT COUNT(Title) AS “Number of Books ” FROM Books;** |  |
| **MAX**(*[DISTINCT | ALL] expr*) | This function returns the highest value in expr.   * If expr is date/time value then it returns DATE; the maximum is the latest dates * If expr is numeric then it returns a NUMBER. * If expr is a character value then it returns VARCHAR2 | **SELECT MAX(Price) AS “Maximum Price” FROM Books;** |  |
| **MIN**(*[DISTINCT | ALL] expr*) | This function returns the lowest value in expr.   * If expr is date/time value then it returns DATE, the minimum is the earliest date. * If expr is numeric then it returns a NUMBER. * If expr is a character value then it returns VARCHAR2 | **SELECT MIN(Price) AS “Minimum Price” FROM Books;** |  |
| **STDDEV**(*[DISTINCT | ALL] x*) | This function returns the numeric standard deviation of the expression x. the standard deviation is calculated as the square root of the variance. | **SELECT STDDEV(Price) AS “STD Deviation” FROM Books;** |  |
| **SUM**(*[DISTINCT | ALL] n*) | This function returns the sum of expression n. | **SELECT SUM(Quantity) AS “Total Inventory” FROM Inventory;** |  |
| **VARIANCE** (*DISTINCT | ALL] x*) | This function returns the variance of expression x | **SELECT VARIANCE(Price) AS “Variance Price” FROM Books;** |  |

**GROUP function with DISTINCT keyword**

All the group function can be forced to work on distinct record set, which is done only when keyword distinct is mentioned with the group function. The example 5.1 below shows the use of DISTINCT keyword in a SELECT statement with group function.

|  |  |  |
| --- | --- | --- |
| **Example 5.1: Using DISTINCT keyword with GROUP Function**   |  | | --- | | **SELECT COUNT(DISTINCT P\_Code) AS “Distinct Publishers” FROM Books;** | |  | |
|  |

Here in this example the Oracle calculates the distinct Publishers for the books in the Books table eliminating all duplicates. By default also it will count the number of Publishers but then it will calculate the duplicate the COUNT value as two books can have one Publisher.

**GROUP BY**

The GROUP BY clause is a part of SQL SELECT statement and is used in conjunction with the aggregate functions to group the result-set by one or more columns. When the query with GROUP BY clause executes and the data is fetched, it is grouped based on the GROUP BY clause and the group by function is applied.

In short GROUP BY is used to divide the big table information into smaller groups. The example 5.2 below is used to demonstrate the use of GROUP BY clause.

|  |  |  |
| --- | --- | --- |
| **Example 5.2: Using GROUP BY clause with SELECT statement**   |  | | --- | | **SELECT B\_Code, COUNT(Author\_id) AS “Number of Authors”**  **FROM Written\_By**  **GROUP BY B\_Code;** | |  | |
|  |

When using the GROUP BY clause, we need to keep in mind that all column mentioned in the SELECT list and not included in the group function are mentioned in the GROUP BY clause. Here in this example we are just mentioning B\_Code in the SELECT list so that is also mentioned in the GROUP BY clause, if there are more than one column in the SELECT list then they all are also included in the GROUP BY clause.

While writing a SELECT statement with GROUP BY clause we need to keep some other point in mind as well which are known as guidelines, these are mentioned below:

* We can use WHERE clause we can restrict rows before they are divided into groups.
* Columns must be included in the GROUP BY clause.
* No column alias can be used in the GROUP BY clause.
* By default all rows are sorted in ascending order to invert the order we can use ORDER BY clause.

Example 5.2 shows us the use of GROUP BY clause with single row, it is also necessary to know how to query with multiple columns using the GROUP BY clause.

|  |  |  |
| --- | --- | --- |
| **Example 5.3: Using GROUP BY clause on Multiple Columns**   |  | | --- | | **SELECT Paperback, Type, COUNT(Title) AS “Total Books”**  **FROM Books**  **GROUP BY Paperback, Type;** | |  | |
|  |

Here in this example the GROUP BY clause first groups the data on the basis of Paperback and after that again regroup the data according to the TYPE of the book.

**Restricting the Group results**

Group function cannot be used with the WHERE clause, but sometime we might want to restrict our data when using the group functions. To do so we use the HAVING clause after the GROUP BY which can use the group function to restrict the groups. The syntax of the SELECT statement with HAVING clause is given below:

|  |
| --- |
| **Syntax:**    SELECT [\* | DISTINCT] columnname1 [, columnname2]  FROM tablename1 [, tablename2]  [WHERE condition] [And | or | LIKE condition . . .]  [GROUP BY column-list]  [HAVING “conditions”]  [ORDER BY “condition-list” [ASC | DESC]] |

With this our evaluation of SELECT statement is complete.

The Oracle server performs the following steps when the HAVING clause is used:

1. Rows are grouped.
2. The group function is applied to the group
3. The groups that match the criteria in the HAVING clause are displayed.

We can use HAVING clause before the GROUP BY clause but it is always recommended to use the GROUP BY clause first

Example 5.4 below now demonstrates the use of HAVING in SELECT statement, we have taken here the same example as in 5.2 but just added a HAVING clause to it which restrict the records to be displayed.

|  |  |  |
| --- | --- | --- |
| **Example 5.4: Using the HAVING clause in the SELECT statement**   |  | | --- | | **SELECT Paperback, Type, COUNT(Title) AS “Total Books”**  **FROM Books**  **GROUP BY Paperback, Type**  **HAVING COUNT(Title) > 2;** | |  | |
|  |

**Nesting Group Functions**

The group function can be nested to depth of two; the example 5.5 below displays the maximum average price of Books which are grouped by the P\_Code.

|  |  |  |
| --- | --- | --- |
| **Example 5.5: Nesting Group Functions in SELECT statement**   |  | | --- | | **SELECT MAX(AVG(Price))**  **FROM Books**  **GROUP BY P\_Code;** | |  | |
|  |

We always need to keep few points in mind that always place HAVING and GROUP BY clause after the WHERE clause in the SELECT statement and always place ORDER BY clause at the end.

The Oracle server always follows the following sequence to evaluate the SELECT statement:

1. If the WHERE clause exist in the statement, the server extracts the candidate rows.
2. Then the server identifies the groups specified in the GROUP BY clause.
3. Further the HAVING clause restrict the result set based on the grouping criteria mentioned in the HAVING clause.

## SOLVED PRACTICE QUESTIONS

**Practice Set – 5.1**

1. **Group functions work across many rows to produce one result per group.**

**True/False**

**Solution:**

**True**

1. **Group functions include NULLs in calculations.**

**True/False**

**Solution:**

**True**

1. **Display the details of book with highest price.**

**Solution:**

**SELECT MAX(Price) FROM Books**

**Output:**



1. **Display the details of books with lowest price.**

**Solution:**

**SELECT MIN(Price) FROM Books;**

**Output:**



1. **Display the average price of all the books in BOOKS table.**

**Solution:**

**SELECT AVG(Price) FROM Books;**

**Output:**



1. **Display the sum of price for each type of Book.**

**Solution:**

**SELECT SUM (Price) From Books;**

**Output:**



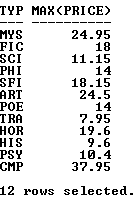
1. **Display the maximum price for each type of book.**

**Solution:**

**SELECT type, MAX(Price) FROM Books**

**GROUP BY Type;**

**Output:**



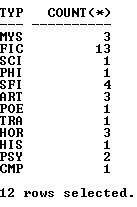
1. **Display the number of books of similar type.**

**Solution:**

**SELECT Type, COUNT(\*) FROM Books**

**GROUP BY Type;**

**Output:**



1. **Write a query to determine the difference between the highest and lowest price of the books.**

**Solution:**

**SELECT ((MAX (Price)) – (MIN (Price))) AS “Difference”**

**FROM Books;**

**Output:**



## UNSOLVED PRACTICE QUESTIONS

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

**Practice Set – 5.2**

1. **Display the highest, lowest, sum and average prices of all products. Label the columns as Maximum, Minimum, Sum, and Average respectively.**
2. **Display the maximum, minimum, sum and average of prices of products in each department.**
3. **Display the number of products delivered by each Supplier.**
4. **Determine the total number of product delivered in each department.**
5. **What is the difference of price between the highest and the lowest product price for each department?**
6. **Display the department name and the name of the cheapest product delivered in the department.**
7. **Display the name and price of the most expensive product delivered by S6.**
8. **Display the list of suppliers and the cheapest product for that supplier. Sort the result in the descending order of Price.**
9. **Display the department name, supplier name and the average price for all the products delivered in that department.**
10. **Display the supplier Id with the maximum numbers of deliveries.**

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