DATA BASE MANAGEMENT SYSTEMS

Unit – IV

Relational Database Language: Data definition in SQL, Queries in SQL, Insert, Delete and Update Statements in SQL, Views in SQL, Specifying General Constraints as Assertions, specifying indexes, Embedded SQL. PL /SQL: Introduction .

Unit-IV

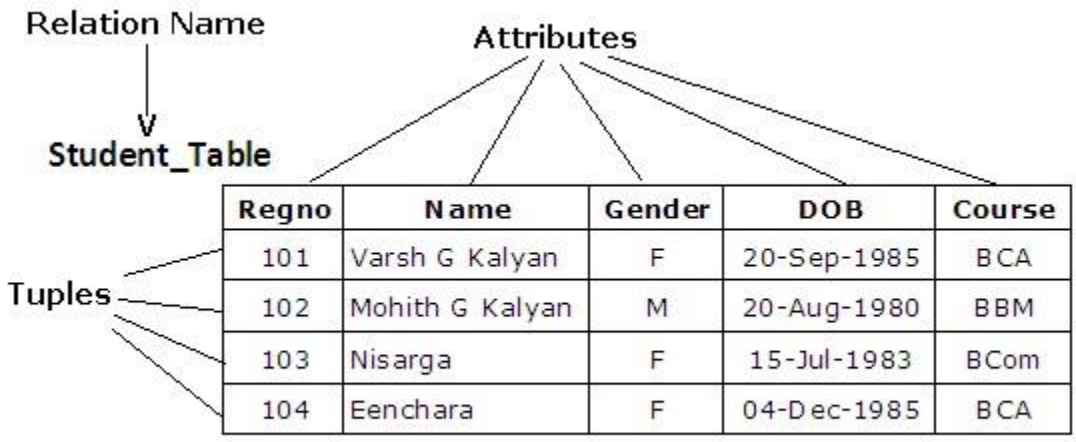
Relational Database Language

SQL

- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases
- SQL is an ANSI (American National Standards Institute) standard

SQL is a declarative (non-procedural)language. SQL is (usually) not case-sensitive, but we'll write SQL keywords in upper case for emphasis.
Some database systems require a semicolon at the end of each SQL statement.

A table is database object that holds user data. Each column of the table will have specified data type bound to it. Oracle ensures that only data, which is identical to the datatype of the column, will be stored within the column.



Regno	Name	Gender	DOB	Course
101	Varsh G Kalyan	F	20-Sep-1985	BCA
102	Mohith G Kalyan	M	20-Aug-1980	BBM
103	Nisarga	F	15-Jul-1983	BCom
104	Eenchara	F	04-Dec-1985	BCA

Fig: The attributes and tuples of a relation STUDENT_Table

SQL DML and DDL

SQL can be divided into two parts:

The Data Definition Language (DDL) and the Data Manipulation Language (DML).

Data Definition Language (DDL)

It is a set of SQL commands used to create, modify and delete database structure but not data. It also define indexes (keys), specify links between tables, and impose constraints between tables. DDL commands are auto COMMIT.

The most important DDL statements in SQL are:

- CREATE TABLE - creates a new table
- ALTER TABLE - modifies a table
- TRUNCATE TABLE- deletes all records from a table
- DROP TABLE - deletes a table

Data Manipulation Language (DML)

It is the area of SQL that allows changing data within the database. The query and update commands form the DML part of SQL:

- INSERT - inserts new data into a database
- SELECT - extracts data from a database
- UPDATE - updates data in a database
- DELETE - deletes data from a database

Data Control Language (DCL)

It is the component of SQL statement that control access to data and to the database. Occasionally DCL statements are grouped with DML Statements.

- COMMIT –Save work done.
- SAVEPOINT – Identify a point in a transaction to which you can later rollback.
- ROLLBACK – Restore database to original since the last COMMIT.
- GRANT – gives user's access privileges to database.
- REVOKE – withdraw access privileges given with GRANT command.

Basic Data Types

Data Type	Description
CHAR(size)	This data type is used to store character strings values of fixed length. The size in brackets determines the number of characters the cell can hold. The maximum number of character(ie the size) this data type can hold is 255 characters. The data held is right padded with spaces to whatever length specified.
VARCHAR(size) / VARCHAR2(size)	This data type is used to store variable length alphanumeric data. It is more flexible form of CHAR data type. VARCHAR can hold 1 to 255 characters. VARCHAR is usually a wiser choice than CHAR, due to its variable length format characteristic. But, keep in mind, that CHAR is much faster than VARCHAR, sometimes up to 50%.
DATE	This data type is used to represent data and time. The standard format is DD-MMM-YY. Date Time stores date in the 24-hour format. By default, the time in a date field is 12:00:00am.
NUMBER(P,S)	The NUMBER data type is used to store numbers(fixed or floating point). Number of virtually any magnitude maybe stored up to 38 digits of precision. The Precision(P), determines the maximum length of the data, whereas the scale(S), determine the number of places to the right of the decimal. Example: Number(5,2) is a number that has 3 digits before the decimal and 2 digits after the decimal.
LONG	This data type is used to store variable length character strings containing up to 2GB. LONG data can be used to store arrays of binary data in ASCII format. Only one LONG value can be defined per table.

RAW / LONG RAW	The RAW / LONG RAW data types are used to store binary data, such as digitized picture or image. RAW data type can have maximum length of 255 bytes. LONG RAW data type can contain up to 2GB.

The CREATE TABLE Command:

The CREATE TABLE command defines each column of the table uniquely. Each column has a minimum of three attributes, name, datatype and size(i.e column width).each table column definition is a single clause in the create table syntax. Each table column definition is separated from the other by a comma. Finally, the SQL statement is terminated with a semi colon.

Rules for Creating Tables

A name can have maximum upto 30 characters.

Alphabets from A-Z, a-z and numbers from 0-9 are allowed.

A name should begin with an alphabet.

The use of the special character like _(underscore) is allowed.

SQL reserved words not allowed. For example: create, select, alter.

Syntax:

CREATE TABLE <tablename>
(<columnName1> <Datatype>(<size>),
<columnName2> <Datatype>(<size>), …….);

Example:

```
CREATE TABLE gktab
(Regno NUMBER(3),
Name VARCHAR(20),
Gender CHAR,
Dob DATE,
Course CHAR(5));
```

Inserting Data into Tables

Once a table is created, the most natural thing to do is load this table with data to be manipulated later.

When inserting a single row of data into the table, the insert operation:

Creates a new row(empty) in the database table.
Loads the values passed(by the SQL insert) into the columns specified.

Syntax:

INSERT INTO <tablename>(<columnname1>, <columnname2>, ..)

Values(<expression1>,<expression2>…);

Example:

```
INSERT INTO gktab(regno,name,gender,dob,course)
    VALUES(101,'Varsh G Kalyan','F','20-Sep-1985','BCA');
```

Or you can use the below method to insert the data into table.

```
INSERT INTO gktab VALUES(102,'Mohith G Kalyan','M','20-Aug-1980','BBM');
INSERT INTO gktab VALUES(106,'Nisarga','F','15-Jul-1983','BCom');
INSERT INTO gktab VALUES(105,'Eenchara','F','04-Dec-1985','BCA');
INSERT INTO gktab VALUES(103,'Ravi K','M','29-Mar-1989','BCom');
INSERT INTO gktab VALUES(104,'Roopa','F','17-Jan-1984','BBM');
```

Whenever you work on the data which has data types like
CHAR,VARCHAR/VARCHAR2, DATE should be used between single quote(‘)

Viewing Data in the Tables

Once data has been inserted into a table, the next most logical operation would be to view what has been inserted. The **SELECT** SQL verb is used to achieve this. The SELECT command is used to retrieve rows selected from one or more tables.

All Rows and All Columns

SELECT * FROM <tablename>

SELECT * FROM gktab;

It shows all rows and column data in the table

Regno	Name	Gender	Dob	Course
101	Varsh G Kalyan	F	20-Sep-1985	BCA
102	Mohith G Kalyan	M	20-Aug-1980	BBM
106	Nisarga	F	15-Jul-1983	BCom
105	Eenchara	F	04-Dec-1985	BCA
103	Ravi K	M	29-Mar-1989	BCom
104	Roopa	F	17-Jan-1984	BBM

Filtering Table Data

While viewing data from a table it is rare that all the data from the table will be required each time. Hence, SQL provides a method of filtering table data that is not required.

The ways of filtering table data are:

Selected columns and all rows
Selected rows and all columns
Selected columns and selected rows

Selected Columns and All Rows

The retrieval of specific columns from a table can be done as shown below.

Syntax

SELECT <columnname1>, <Columnname2> FROM <tablename>

Example

Show only Regno, Name and Course from gktab.

SELECT Regno, Name, Course FROM gktab;

Regno	Name	Course
101	Varsh G Kalyan	BCA
102	Mohith G Kalyan	BBM
106	Nisarga	BCom
105	Eenchara	BCA
103	Ravi K	BCom
104	Roopa	BBM

Selected Rows and All Columns

The WHERE clause is used to extract only those records that fulfill a specified criterion.

When a WHERE clause is added to the SQL query, the Oracle engine compares each record in the table with condition specified in the WHERE clause. The Oracle engine displays only those records that satisfy the specified condition.

Syntax

SELECT * FROM <tablename> WHERE <condition>;

Here, ***<condition>*** is always quantified as ***<columnname=value>***

When specifying a condition in the **WHERE** clause all standard operators such as logical, arithmetic and so on, can be used.

Example-1:

Display all the students from BCA.

SELECT * FROM gktab WHERE Course='BCA';

Regno	Name	Gender	Dob	Course
101	Varsh G Kalyan	F	20-Sep-1985	BCA
105	Eenchara	F	04-Dec-1985	BCA

Example-2:

Display the student whose regno is 102.

SELECT * FROM gktab WHERE Regno=102;

Regno	Name	Gender	Dob	Course
102	Mohith G Kalyan	M	20-Aug-1980	BBM

Selected Columns and Selected Rows

To view a specific set of rows and columns from a table

When a WHERE clause is added to the SQL query, the Oracle engine compares each record in the table with condition specified in the WHERE clause. The Oracle engine displays only those records that satisfy the specified condition.

Syntax

SELECT <columnname1>, <Columnname2> FROM <tablename>

WHERE <condition>;

Example-1:

List the student's Regno, Name for the Course BCA.

SELECT Regno, Name FROM gktab WHERE Course='BCA';

Regno	Name
101	Varsh G Kalyan
105	Eenchara

Example-2:

List the student's Regno, Name, Gender for the Course BBM.

SELECT Regno, Name, Gender FROM gktab WHERE Course='BBM';

Regno	Name	Gender
102	Mohith G Kalyan	M
104	Roopa	F

Eliminating Duplicate Rows when using a SELECT statement

A table could hold duplicate rows. In such a case, to view only unique rows the **DISTINCT** clause can be used.

The **DISTINCT** clause allows removing duplicates from the result set. The **DISTINCT** clause can only be used with **SELECT** statements.

The **DISTINCT** clause scans through the values of the column/s specified and displays only unique values from amongst them.

Syntax

SELECT DISTINCT <columnname1>, <Columnname2>

FROM <Tablename>;

Example:

Show different courses from gktab

SELECT DISTINCT Course from gktab;

Course
BCA
BBM
BCom

Sorting Data in a Table

Oracle allows data from a table to be viewed in a sorted order. The rows retrieved from the table will be sorted in either ***ascending*** or ***descending*** order depending on the condition specified in the ***SELECT*** sentence.

Syntax

SELECT * FROM <tablename>

ORDER BY <Columnname1>,<Columnname2> <[Sort Order]>;

The ***ORDER BY*** clause sorts the result set based on the column specified. The

ORDER BY clause can only be used in ***SELECT*** statements.

The Oracle engine sorts in ***ascending order by default***

Example-1:

Show details of students according to Regno.

SELECT * FROM gktab ORDER BY Regno;

Regno	Name	Gender	Dob	Course
101	Varsh G Kalyan	F	20-Sep-1985	BCA
102	Mohith G Kalyan	M	20-Aug-1980	BBM
103	Ravi K	M	29-Mar-1989	BCom
104	Roopa	F	17-Jan-1984	BBM
105	Eenchara	F	04-Dec-1985	BCA
106	Nisarga	F	15-Jul-1983	BCom

Regno Sorted

Example-2:

Show the details of students names in descending order.

SELECT * FROM gktab ORDER BY Name DESC;

Regno	Name	Gender	Dob	Course
101	Varsh G Kalyan	F	20-Sep-1985	BCA
104	Roopa	F	17-Jan-1984	BBM
103	Ravi K	M	29-Mar-1989	BCom
106	Nisarga	F	15-Jul-1983	BCom
102	Mohith G Kalyan	M	20-Aug-1980	BBM
105	Eenchara	F	04-Dec-1985	BCA

Name Sorted in descending order

DELETE Operations

The ***DELETE*** command deletes rows from the table that satisfies the condition provided by its ***WHERE*** clause, and returns the number of records deleted.

The verb ***DELETE*** in SQL is used to remove either

Specific row(s) from a table

OR

All the rows from a table

Removal of Specific Row(s)

Syntax:

DELETE FROM tablename WHERE Condition;

Example:

DELETE FROM gktab WHERE Regno=103;

1 rows deleted

SELECT * FROM gktab;

Regno	Name	Gender	Dob	Course
101	Varsh G Kalyan	F	20-Sep-1985	BCA
102	Mohith G Kalyan	M	20-Aug-1980	BBM
106	Nisarga	F	15-Jul-1983	BCom
105	Eenchara	F	04-Dec-1985	BCA
104	Roopa	F	17-Jan-1984	BBM

In the above table, the Regno 103 is deleted from the table

Remove of ALL Rows

Syntax

DELETE FROM tablename;

Example

DELETE FROM gktab;

6 rows deleted

SELECT * FROM gktab;

no rows selected

Once the table is deleted, use Rollback to undo the above operations.

UPDATING THE CONTENTS OF A TABLE

The **UPDATE** Command is used to change or modify data values in a table.

The verb update in SQL is used to either updates:

ALL the rows from a table.

OR

A select set of rows from a table.

Updating all rows

The **UPDATE** statement updates columns in the existing table's rows with a new values. The **SET** clause indicates which column data should be modified and the new values that they should hold. The WHERE clause, if given, specifies which rows should be updated. Otherwise, all table rows are updated.

Syntax:

UPDATE tablename

SET columnname1=expression1, columnname2=expression2;

Example: update the gktab table by changing its course to BCA.

UPDATE gktab SET course='BCA';

6 rows updated

SELECT * FROM gktab;

Regno	Name	Gender	Dob	Course
101	Varsh G Kalyan	F	20-Sep-1985	BCA
102	Mohith G Kalyan	M	20-Aug-1980	BCA
106	Nisarga	F	15-Jul-1983	BCA
105	Eenchara	F	04-Dec-1985	BCA
103	Ravi K	M	29-Mar-1989	BCA
104	Roopa	F	17-Jan-1984	BCA

In the above table, the course is changed to BCA for all the rows in the table.

Updating Records Conditionally

If you want to update a specific set of rows in table, then ***WHERE*** clause is used.

Syntax:

UPDATE tablename

SET Columnname1=Expression1, Columnname2=Expression2

WHERE Condition;

Example:

Update gktab table by changing the course BCA to BBM for Regno 102.

UPDATE gktab SET Course='BBM' WHERE Regno=102;

1 rows updated

SELECT * FROM gktab;

Regno	Name	Gender	Dob	Course
101	Varsh G Kalyan	F	20-Sep-1985	BCA
102	Mohith G Kalyan	M	20-Aug-1980	BBM
106	Nisarga	F	15-Jul-1983	BCA
105	Eenchara	F	04-Dec-1985	BCA
103	Ravi K	M	29-Mar-1989	BCA
104	Roopa	F	17-Jan-1984	BCA

MODIFYING THE STRUCTURE OF TABLES

The structure of a table can be modified by using the **ALTER TABLE** command.

ALTER TABLE allows changing the structure of an existing table. With ***ALTER TABLE*** if is possible to ***add*** or ***delete*** columns, ***create*** or ***destroy*** indexes, ***change the data type*** of existing columns, or ***rename columns*** or the ***table*** itself.

ALTER TABLE works by making a temporary copy of the original table. The alteration is performed on the copy, then the original table is deleted and the new one is renamed. While ***ALTER TABLE*** is executing, the original table is still readable by the users of ORACLE.

Restrictions on the ALTER TABLE

The following task **cannot** be performed when using the ***ALTER TABLE*** Clause:

Change the name of the table.

Change the name of the Column.

Decrease the size of a column if table data exists.

ALTER TABLE Command can perform

Adding New Columns.

Dropping A Column from a Table.

Modifying Existing Columns.

Adding New Columns

Syntax:

```
ALTER TABLE tablename
        ADD(NewColumnname1    Datatype(size),
            NewColumnname2    Datatype(size).....);
```

Example: Enter a new filed Phno to gktab.

ALTER TABLE gktab ***ADD(***Phno number(10)***);***

The table is altered with new column Phno

Select * from gktab;

Regno	Name	Gender	Dob	Course	Phno
101	Varsh G Kalyan	F	20-Sep-1985	BCA	
102	Mohith G Kalyan	M	20-Aug-1980	BBM	
106	Nisarga	F	15-Jul-1983	BCom	
105	Eenchara	F	04-Dec-1985	BCA	
103	Ravi K	M	29-Mar-1989	BCom	
104	Roopa	F	17-Jan-1984	BBM	

You can also use ***DESC*** gktab, to see the new column added to table.

Dropping A Column from a Table.

Syntax:

ALTER TABLE tablename DROP COLUMN Columnname;

Example: Drop the column Phno from gktab.

ALTER TABLE gktab ***DROP COLUMN*** Phno***;***

The table is altered, the column Phno is removed from the table.

Select * from gktab;

Regno	Name	Gender	Dob	Course
101	Varsh G Kalyan	F	20-Sep-1985	BCA
102	Mohith G Kalyan	M	20-Aug-1980	BBM
106	Nisarga	F	15-Jul-1983	BCom
105	Eenchara	F	04-Dec-1985	BCA
103	Ravi K	M	29-Mar-1989	BCom
104	Roopa	F	17-Jan-1984	BBM

You can also use ***DESC*** gktab, to see the column removed from the table.

Modifying Existing Columns.

Syntax:

ALTER TABLE tablename

MODIFY(Columnname Newdatatype(Newsize));

Example:

ALTER TABLE gktab ***MODIFY(***Name VARCHAR(25)***);***

The table altered with new size value 25.

DESC gktab;

RENAMING TABLES

Oracle allows renaming of tables. The rename operation is done atomically, which means that no other thread can access any of the tables while the rename process is running.

Syntax

RENAME tablename ***TO*** newtablename***;***

TRUNCATING TABLES

TRUNCATE command deletes the rows in the table permanently.

Syntax:

TRUNCATE TABLE tablename***;***

The number of deleted rows are not returned. Truncate operations drop and re-create the table, which is much faster than deleting rows one by one.

Example:

TRUNCATE TABLE gktab***;***

Table truncated i.e., all the rows are deleted permanently.

DESTROYING TABLES

Sometimes tables within a particular database become obsolete and need to be discarded. In such situation using the **DROP TABLE** statement with table name can destroy a specific table.

Syntax:

DROP TABLE tablename***;***

Example:

DROP TABLE gktab***;***

If a table is dropped all the records held within and the structure of the table is lost and cannot be recovered.

COMMIT and ROLLBACK

Commit

Commit command is used to permanently save any transaction into database.

```
SQL> commit;
```

Rollback

Rollback is used to undo the changes made by any command but only before a commit is done. We can't Rollback data which has been committed in the database with the help of the commit keyword or DDL Commands, because DDL commands are auto commit commands.

```
SQL> Rollback;
```

Difference between DELETE and DROP.

The **DELETE** command is used to remove rows from a table. After performing a DELETE operation you need to COMMIT or ROLLBACK the transaction to make the change permanent or to undo it.

The **DROP** command removes a table from the database. All the tables' rows, indexes and privileges will also be removed. The operation cannot be rolled back.

Difference between DELETE and TRUNCATE.

The **DELETE** command is used to remove rows from a table. After performing a DELETE operation you need to COMMIT or ROLLBACK the transaction to make the change permanent or to undo it.

TRUNCATE removes all rows from a table. The operation cannot be rolled back.

Difference between CHAR and VARCHAR.

CHAR

1. Used to store fixed length data.
2. The maximum characters the data type can hold is 255 characters.
3. It's 50% faster than VARCHAR.
4. Uses static memory allocation.

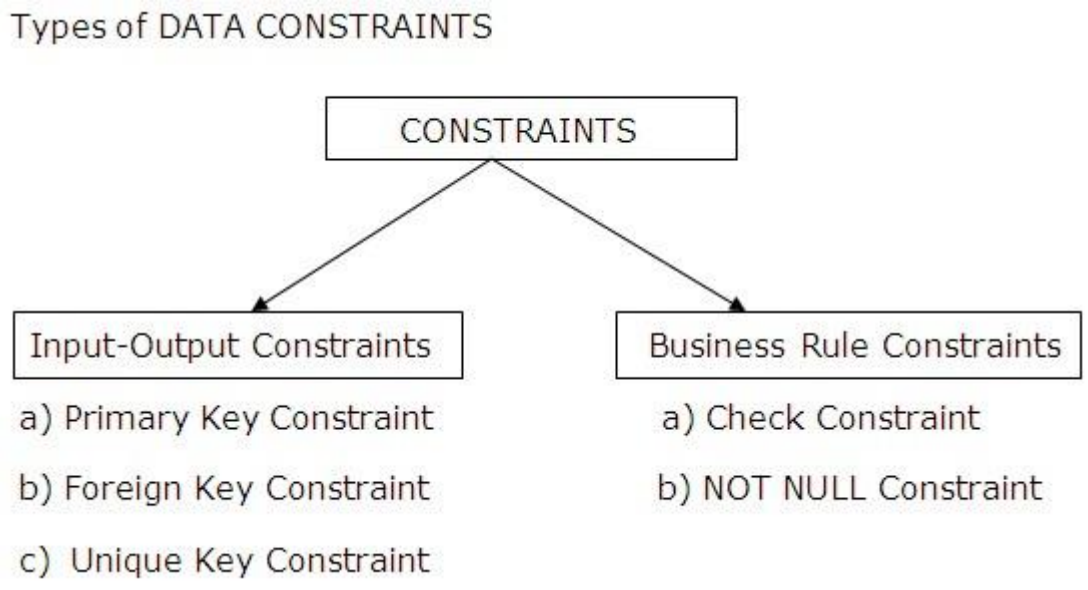
VARCHAR

1. Used to store variable length data.
2. The maximum characters the data type can hold is up to 4000 characters.
3. It's slower than CHAR.
4. Uses dynamic memory allocation.

DATA CONSTRINTS

Oracle permits data constraints to be attached to table column via SQL syntax that checks data for integrity prior storage. Once data constraints are part of a table column construct, the oracle database engine checks the data being entered into a table column against the data constraints. If the data passes this check, it is stored in the table column, else the data is rejected. Even if a single column of the record being entered into the table fails a constraint, the entire record is rejected and not stored in the table.

Both **CREATE TABLE** and **ALTER TABLE** SQL verbs can be used to write SQL sentences that attach constraints to a table column.
The constraints are a keyword. The constraint is rules that restrict the values for one or more columns in a table. The Oracle Server uses constraints to prevent invalid data entry into tables. The constraints store the validate data and without constraints we can just store invalid data. The constraints are an important part of the table.



Primary Key Constraint

A primary key can consist of one or more columns on a table. Primary key constraints define a column or series of columns that uniquely identify a given row in a table. Defining a primary key on a table is optional and you can only define a single primary key on a table. A primary key constraint can consist of one or many columns (up to 32). When multiple columns are used as a primary key, they are called a composite key. Any column that is defined as a primary key column is automatically set with a NOT NULL status. The Primary key constraint can be applied at column level and table level.

Foreign Key Constraint

A foreign key constraint is used to enforce a relationship between two tables. A foreign key is a column (or a group of columns) whose values are derived from the Primary key or unique key of some other table.

The table in which the foreign key is defined is called a Foreign table or Detail table. The table that defines primary key or unique key and is referenced by the foreign key is called Primary table or Master table.

The master table can be referenced in the foreign key definition by using the clause **REFERENCES Tablename.ColumnName** when defining the foreign key, column attributes, in the detail table. The foreign key constraint can be applied at column level and table level.

Unique Key Constraint

Unique key will not allow duplicate values. A table can have more than one Unique key. A unique constraint defines a column, or series of columns, that must be unique in value. The UNIQUE constraint can be applied at column level and table level.

CHECK Constraint

Business Rule validation can be applied to a table column by using **CHECk** constraint. **CHECK** constraints must be specified as a logical expression that evaluates either to **TRUE** or **FALSE**.

The **CHECK** constraint ensures that all values in a column satisfy certain conditions. Once defined, the database will only insert a new row or update an existing row if the new value satisfies the **CHECK** constraint. The **CHECK** constraint is used to ensure data quality.

A CHECK constraint takes substantially longer to execute as compared to NOT NULL, PRIMARY KEY, FOREIGN KEY or UNIQUE. The CHECK constraint can be applied at column level and table level.

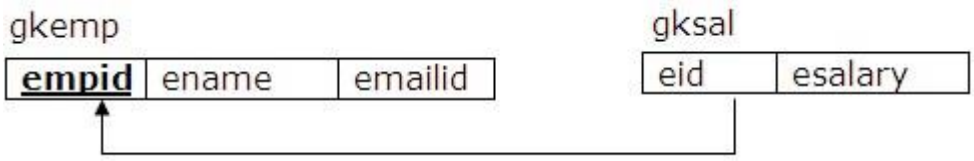
NOT NULL Constraint

The NOT NULL column constraint ensures that a table column cannot be left empty.

When a column is defined as not null, then that column becomes a mandatory column. The NOT NULL constraint can only be applied at column level.

Example on Constraints

Consider the Table shown below




```
SQL> create table gkemp(empid number(3) primary key,
  2  ename varchar(20) not null,
  3  emailid varchar(15) unique);

Table created.

SQL> create table gksal(eid number(3) references gkemp(empid),
  2  esal number(5) check(esal between 5000 and 90000));

Table created.
```

Arithmetic Operators

Oracle allows arithmetic operators to be used while viewing records from a table or while performing data manipulation operations such as insert, updated and delete. These are:

+ Addition
- Subtraction
/ Division
* Multiplication
() Enclosed Operations

Consider the below employee table(gkemp)

```
SQL> select * from gkemp;

     EMPID ENAME                          ESAL
---------- -------------------- ----------
       101 Nisarga                       8500
       102 Varsha G Kalyan              15000
       103 Eenchara                      5000
       104 Mohith G Kalyan              12500
       105 Kavitha                      18000
```

```
SQL> select esal,esal+2500 from gkemp;

      ESAL  ESAL+2500
---------- ----------
      8500       11000
     15000       17500
      5000        7500
     12500       15000
     18000       20500

SQL> select esal,esal-500 from gkemp;

      ESAL   ESAL-500
---------- ----------
      8500        8000
     15000       14500
      5000        4500
     12500       12000
     18000       17500

SQL> select esal, esal/100 from gkemp;

      ESAL   ESAL/100
---------- ----------
      8500          85
     15000         150
      5000          50
     12500         125
     18000         180

SQL> select esal, esal*10 from gkemp;

      ESAL    ESAL*10
---------- ----------
      8500       85000
     15000      150000
      5000       50000
     12500      125000
     18000      180000
SQL>  select esal,(10+esal)/100 from gkemp;

     ESAL (10+ESAL)/100
--------- -------------
     8500          85.1
    15000         150.1
     5000          50.1
    12500         125.1
    18000         180.1
```

Special Note

The **DUAL** table is a special one-row, one-column table present by default in Oracle and other database installations. Dual is a dummy table.

```
SQL> select (100+1560) from dual;

(100+1560)
----------
      1660

SQL> select (250-34) from dual;

  (250-34)
----------
       216

SQL> select (24*10) from dual;

   (24*10)
----------
       240

SQL> select (24+32/4-24) from dual;

(24+32/4-24)
------------
           8
```

Logical Operators

Logical operators that can be used in SQL sentence are:

AND Operators
OR Operators
NOT Operators

Operators ***Description***

OR :-For the row to be selected at least one of the conditions must be true.

AND :-For a row to be selected all the specified conditions must be true.

NOT :-For a row to be selected the specified condition must be false.

Consider the below employee table(gkemp)

```
SQL> select * from gkemp;

     EMPID ENAME                      ESAL DEPARTMENT
---------- -------------------- ---------- ------------------
       101 Nisarga                    8500 Commerce
       102 Varsha G Kalyan           15000 Computer Science
       103 Eenchara                   5000 Commerce
       104 Mohith G Kalyan           12500 Computer Science
       105 Kavitha                   18000 Arts
```

For example: if you want to find the names of employees who are working either in Commerce or Arts department, the query would be like,

```
SQL>  select * from gkemp
  2  where department='Commerce' or department='Arts';

     EMPID ENAME                        ESAL DEPARTMENT
---------- -------------------- ---------- ------------
       101 Nisarga                    8500 Commerce
       103 Eenchara                   5000 Commerce
       105 Kavitha                   18000 Arts
```

For example: To find the names of the employee whose salary between10000 to 20000, the query would be like,

```
SQL> select * from gkemp
  2   where esal>=10000  and esal<=20000;

     EMPID ENAME                        ESAL DEPARTMENT
---------- -------------------- ---------- ------------------
       102 Varsha G Kalyan           15000 Computer Science
       104 Mohith G Kalyan           12500 Computer Science
       105 Kavitha                   18000 Arts
```

For example: If you want to find out the names of the employee who do not belong to computer science department, the query would be like,

```
SQL>  select * from gkemp
  2   where not department='Computer Science';

     EMPID ENAME                          ESAL DEPARTMENT
---------- -------------------- ---------- ----------
       101 Nisarga                      8500 Commerce
       103 Eenchara                     5000 Commerce
       105 Kavitha                     18000 Arts
```

Range Searching (BETWEEN)

In order to select the data that is within a range of values, the **BETWEEN** operator is used. The BETWEEN operator allows the selection of rows that contain values within a specified lower and upper limit. The range coded after the word BETWEEN is inclusive.

The lower value must be coded first. The two values in between the range must be linked with the keyword **AND**. The BETWEEN operator can be used with both character and numeric data types. However, the data types cannot be mixed.

For example: Find the names of the employee whose salary between10000 and 20000, the query would be like,

```
SQL> select * from gkemp
  2  where esal between 10000 and 20000;

     EMPID ENAME                          ESAL DEPARTMENT
---------- -------------------- ---------- ------------------
       102 Varsha G Kalyan           15000 Computer Science
       104 Mohith G Kalyan           12500 Computer Science
       105 Kavitha                   18000 Arts
```

Pattern Matching (LIKE, IN, NOT IN)

LIKE

The **LIKE** predicate allows comparison of one string value with another string value, which is not identical. this is achieved by using wild characters. Two wild characters that are available are:

For character data types:

% allows to match any string of any length (including zero length).

_ allows to match on a single character.

```
SQL> select * from company;

COMPANY_ID COMPANY_NAME               COMPANY_CITY
---------- ------------------------- --------------
       801 Aarti Industries          Mumbai
       802 ABB India Ltd             Bangalore
       803 Adani Power Ltd           Ahmedbad
       804 Balmer Lawrie             Kolkata
       805 Biocon Ltd                Bangalore
       806 Minda Industries Ltd      Gurgaon

SQL> select * from company where company_name like 'A%';

COMPANY_ID COMPANY_NAME               COMPANY_CITY
---------- ------------------------- -------------------------
       801 Aarti Industries          Mumbai
       802 ABB India Ltd             Bangalore
       803 Adani Power Ltd           Ahmedbad

SQL> select * from company where COMPANY_CITY like '_u%';

COMPANY_ID COMPANY_NAME               COMPANY_CITY
---------- ------------------------- -------------------------
       801 Aarti Industries          Mumbai
       806 Minda Industries Ltd      Gurgaon
```

IN

The IN operator is used when you want to compare a column with more than one value. It is similar to an OR condition.

For example: If you want to find the names of company located in the city Bangalore, Mumbai, Gurgaon, the query would be like,

```
SQL> select * from company
  2  where company_city in('Bangalore','Mumbai','Gurgaon');

COMPANY_ID COMPANY_NAME               COMPANY_CITY
---------- ------------------------- ------------------------
       801 Aarti Industries          Mumbai
       802 ABB India Ltd             Bangalore
       805 Biocon Ltd                Bangalore
       806 Minda Industries Ltd      Gurgaon
```

NOT IN

The NOT IN operator is opposite to IN.

For example: If you want to find the names of company located in the other city of Bangalore, Mumbai, Gurgaon, the query would be like,

```
SQL>  select * from company
  2  where company_city not in('Bangalore','Mumbai','Gurgaon');

COMPANY_ID COMPANY_NAME               COMPANY_CITY
---------- ------------------------- ------------------------
       803 Adani Power Ltd           Ahmedbad
       804 Balmer Lawrie             Kolkata
```

Column Aliases(Renaming Columns) in Oracle:

Sometimes you want to change the column headers in the report. For this you can use **column aliases in oracle**. Oracle has provided excellent object oriented techniques as its robust database. It always good to practice and implement column aliases since it will make your code readable while using this columns.

to add **column aliases** to your sql queries.

Give a **column alias** name separated by space after the column name.

Select DOB **DateofBirth** from gkstudent

In the above query, the word in bold is column aliases.

ORACLE FUNCTIONS

Oracle functions serve the purpose of manipulating data items and returning a result. Functions are also capable of accepting user-supplied variables or constants and operating on them. Such variables or constants are called arguments. Any number of arguments(or no arguments at all) can be passed to a function in the following format.

Function_Name(arguments1,arguments2.....)

Oracle functions can be clubbed together depending upon whether they operate on a single row or a group of rows retrieved from a table. Accordingly, functions can be classified as follows:

Group Functions(Aggregate Functions)

Function that act on a set of values are called group functions.

Scalar Functions(Single Row Functions)

Function that act on only one value at a time are called scalar functions.

String Functions: for string data type

Numeric functions: for Number data type

Conversion function: for conversion of one data type to another.

Date conversions: for date data type.

a) SQL Aggregate / Group Functions

Group functions return results based on groups of rows, rather than on single rows. returns the number of rows in the query.SQL aggregate functions return a single value, calculated from values in a column.

Useful aggregate functions:

a) COUNT() - Returns the number of rows
b) AVG() - Returns the average value
c) MAX() - Returns the largest value
d) MIN() - Returns the smallest value
e) SUM() - Returns the total sum

Consider the below employee table(gkemp)

```
SQL> select * from gkemp;

     EMPID ENAME                          ESAL
---------- -------------------- ----------
       101 Nisarga                       8500
       102 Varsha G Kalyan              15000
       103 Eenchara                      5000
       104 Mohith G Kalyan              12500
       105 Kavitha                      18000
```

a) COUNT()

The COUNT() function counts number of values present in the column excluding Null values.

```
SQL> select count(empid) as totalemployee from gkemp;

TOTALEMPLOYEE
-------------
            5
```

b) AVG()

The AVG() function returns the average value of a column specified.

```
SQL> select avg(esal) as averagesalary from gkemp;

AVERAGESALARY
-------------
        11800
```

c) MAX()

The MAX() function returns the highest value of a particular column.

```
SQL> select max(esal) as maximumsalary from gkemp;

MAXIMUMSALARY
-------------
        18000
```

d) MIN()

The MIN() function returns the smallest value of a particular column.

```
SQL> select min(esal) as minimumsalary from gkemp;

MINIMUMSALARY
-------------
         5000
```

e) SUM()

The SUM() function returns the sum of column values.

```
SQL> select sum(esal) as totalsalary from gkemp;

TOTALSALARY
-----------
      59000
```

b) SQL String Functions

SQL string functions are used primarily for string manipulation. The following table details the important string functions:

SQL Command	Meaning
||	It used for concatenation.
INITCAP	Return a string with first letter of each word in upper case.
LENGTH	Return the length of a word.
LOWER	Returns character, with all letters forced to lowercase.
UPPER	Returns character, with all letters forced to uppercase.
LPAD	Returns character, left-padded to length n with sequence of character specified.
RPAD	Returns character, right-padded to length n with sequence of character specified.
LTRIM	Removes characters from the left of char with initial characters removed upto the first character not in set.
RTRIM	Returns characters, with final characters removed after the last character not in the set.
SUBSTR	Returns a portion of characters, beginning at character **m**, and going upto character **n**. if **n** is omitted, it returns upto the last character in the string. The first position of char is 1.
INSTR	Returns the location of substring in a string.

```
SQL> select ('Ashok') || ('Kumar') as name from dual;

NAME
----------
AshokKumar

SQL> select initcap('nisarga') as name from dual;

NAME
-------
Nisarga

SQL> select length('Varsha G Kalyan') as name from dual;

      NAME
----------
        15

SQL> select lower('PRAKRUTHI') as name from dual;

NAME
---------
prakruthi

SQL> select upper('Computer Science') as name from dual;

NAME
----------------
COMPUTER SCIENCE

SQL> select lpad('sir',4,'r') as name from dual;

NAME
----
rsir

SQL> select rpad('sir',6,'r') as name from dual;

NAME
------
sirrrr
```

```
SQL> select ltrim('Roopa','R') as name from dual;

NAME
----
oopa

SQL> select rtrim('Raman','n') as name from dual;

NAME
----
Rama

SQL> select substr('SECURE',3,4) as name from dual;

NAME
----
CURE

SQL> select instr('Oracle','c') as name from dual;

      NAME
----------
         4
```

Date Conversion Functions

SQL Command	*Meaning*
SYSDATE	It shows the system date.
ADD_MONTHS(d, n)	Returns date after adding the number of months specified in the function.
LAST_DAY(d)	Returns the date of the month specified with the function.
MONTHS_BETWEEN(d1,d2)	Returns number of months between d1 and d2.
NEXT-DAY(date, char)	Returns the date of the first weekday named by char that is after the date named by date. char must be a day of the week
ROUND(date, [format])	Returns a date rounded to a specific unit of measure. If the second parameter is omitted, the ROUND function will round the date to the nearest day.
Conversion Functions	
TO_DATE(<char value>[,<format>])	Converts a character field to a date field
TO-CHAR(<date value>[,<format>])	Converts a date field to a character field

```
SQL> select sysdate "date" from dual;

date
---------
18-JAN-15

SQL> select add_months(sysdate,4) "Add Months" from dual;

Add Month
---------
18-MAY-15

SQL> select sysdate, last_day(sysdate) "LastDay" from dual;

SYSDATE   LastDay
--------- ---------
18-JAN-15 31-JAN-15
```

```
SQL>  select Months_between('02-aug-2015','02-Feb-2015') "months" from dual;

    months
----------
         6

SQL> select Months_between('02-Feb-2015','02-aug-2015') "months" from dual;

    months
----------
        -6

SQL> select next_day('31-jan-2015','Saturday') "Next Day" from dual;

Next Day
---------
07-FEB-15

SQL> select next_day('31-jan-2015', 'Sunday') "Next Day" from dual;

Next Day
---------
01-FEB-15

SQL> select round(to_date('01-jan-2015'),'YYYY') "year" from dual;

year
---------
01-JAN-15

SQL> select round(to_date('01-aug-2015'),'YYYY') "year" from dual;

year
---------
01-JAN-16
```

SET OPERATORS and JOINS

Set Operators

Consider the below tables for set operators examples

```
SQL> create table gkdept(deptno number(2) primary key,
  2  dname varchar(20));

Table created.

SQL>  create table gkemp(empno number(3) primary key,
  2  ename varchar(20),
  3  deptno number(2) references gkdept(deptno));

Table created.

SQL> insert into gkdept values(11,'Computer Science');

1 row created.

SQL> insert into gkdept values(12,'Commerce');

1 row created.

SQL> insert into gkdept values(13,'Management');

1 row created.

SQL> insert into gkdept values(14,'Arts');

1 row created.

SQL> insert into gkemp values(101,'Varsha G Kalyan',11);

1 row created.

SQL> insert into gkemp values(102,'Nisarga',12);

1 row created.

SQL> insert into gkemp values(103,'Eenchara',11);

1 row created.

SQL> insert into gkemp values(104,'Rama',14);

1 row created.
```

```
SQL> select * from gkdept;

    DEPTNO DNAME
---------- --------------------
        11 Computer Science
        12 Commerce
        13 Management
        14 Arts

SQL> select * from gkemp;

     EMPNO ENAME                      DEPTNO
---------- -------------------- ----------
       101 Varsha G Kalyan              11
       102 Nisarga                      12
       103 Eenchara                     11
       104 Rama                         14
```

Set operators combine the result of two quires into single one. The different set operators are:

- **UNION**
- **UNION ALL**
- **INTERSECT**
- **MINUS**

Union Clause

UNION is used to combine the result of two or more SELECT statements. However it will eliminate duplicate rows from its result set. In case of UNION, number of columns in all the query must be same and datatype must be same in both the tables.

Union Example

```
SQL> select deptno from gkdept
  2  union
  3  select deptno from gkemp;

    DEPTNO
----------
        11
        12
        13
        14
```

Union ALL Clause

Same as UNION but it shows the duplicate rows

Union All Example

```
SQL> select deptno from gkdept
  2   union all
  3  select deptno from gkemp;

    DEPTNO
----------
        11
        12
        13
        14
        11
        12
        11
        14
```

Intersect Clause

Intersect is used to combine two SELECT statements, but it only returns the records which are common from both SELECT statement. In case of intersect the number of columns in all the query and datatype must be same.

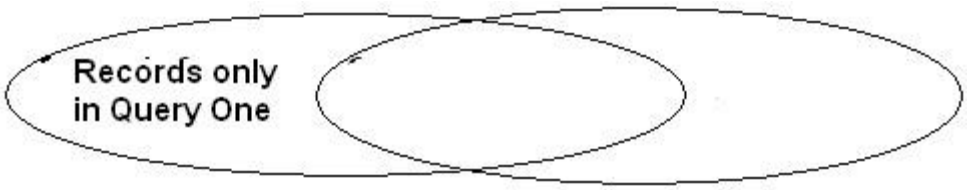
Intersect Example

```
SQL> select deptno from gkdept
  2  intersect
  3  select deptno from gkemp;

    DEPTNO
----------
        11
        12
        14
```

Minus Clause

Minus combines result of two SELECT statement and return only those result which belongs to the first set of result.



Minus Example

```
SQL> select deptno from gkdept
  2  minus
  3  select deptno from gkemp;

    DEPTNO
----------
        13

SQL> select deptno from gkemp
  2  minus
  3  select deptno from gkdept;

no rows selected
```

JOINS

The SQL **Joins** clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.

Here, it is noticeable that the join is performed in the WHERE clause. Several operators can be used to join tables, such as =, <, >, <>, <=, >=, !=, BETWEEN, LIKE, and NOT; they can all be used to join tables. However, the most common operator is the equal symbol.

SQL Join Types:

There are different types of joins available in SQL:

INNER

OUTER(LEFT,RIGHT,FULL)

CROSS

Consider the below tables for Join Operations examples

```
SQL> create table gkproduct(product_id number(3) primary key,
  2  product_name varchar(15),
  3  supplier_name varchar(15),
  4  price number(5));

Table created.

SQL> create table gkorder(order_id number(4) primary key,
  2  product_id number(3) references gkproduct(product_id),
  3  total_units number(3),
  4  customer_name varchar(15));

Table created.
```

```
SQL> insert into gkproduct values(100,'Camera','Nikon',30000);

1 row created.

SQL> insert into gkproduct values(101,'Television','Onida',15000);

1 row created.

SQL> insert into gkproduct values(102,'Refrigerator','videocon',18000);

1 row created.

SQL> insert into gkproduct values(103,'Ipod','Apple',16000);

1 row created.

SQL> insert into gkproduct values(104,'Mobile','Samsung',8000);

1 row created.

SQL> insert into gkorder values(5100,104,30,'Infosys');

1 row created.

SQL> insert into gkorder values(5101,102,15,'GKMV');

1 row created.
SQL> insert into gkorder values(5102,103,25,'Wipro');

1 row created.

SQL>  insert into gkorder values(5103,101,10,'TCS');

1 row created.
```

```
SQL> select * from gkproduct;

PRODUCT_ID PRODUCT_NAME    SUPPLIER_NAME         PRICE
---------- --------------- --------------- ----------
       100 Camera          Nikon                30000
       101 Television      Onida                15000
       102 Refrigerator    videocon             18000
       103 Ipod            Apple                16000
       104 Mobile          Samsung               8000

SQL> select * from gkorder;

  ORDER_ID PRODUCT_ID TOTAL_UNITS CUSTOMER_NAME
---------- ---------- ----------- ---------------
      5100        104          30 Infosys
      5101        102          15 GKMV
      5102        103          25 Wipro
      5103        101          10 TCS
```

INNER Join

Inner join are also known as Equi Joins. They are the most common joins used in SQL. They are known as equi joins because it uses the equal sign as the comparison operator (**=**). The INNER join returns all rows from both tables where there is a match.

Consider the above tables (**gkproduct** and **gkorder**),

For example: If you want to display the product information for each order the query will be as given below

```
SQL> select gko.order_id,gkp.product_name,gkp.price, gkp.supplier_name,gko.total_units
  2  from gkproduct gkp, gkorder gko
  3  where gko.product_id=gkp.product_id;

  ORDER_ID PRODUCT_NAME        PRICE SUPPLIER_NAME   TOTAL_UNITS
---------- --------------- --------- --------------- -----------
      5103 Television         15000 Onida                    10
      5101 Refrigerator       18000 videocon                 15
      5102 Ipod               16000 Apple                    25
      5100 Mobile              8000 Samsung                  30
```

OUTER Join

OUTER join condition returns all rows from both tables which satisfy the join condition along with rows which do not satisfy the join condition from one of the tables. The sql outer join operator in Oracle is (+) and is used on one side of the join condition only.

For example: If you want to display all the product data along with order items data, with null values displayed for order items if a product has no order item, the sql query for outer join would be as shown below(ie First Query).

```
SQL> select gkp.product_id, gkp.product_name,gko.order_id,gko.total_units
  2  from gkproduct gkp, gkorder gko
  3  where gko.product_id(+)=gkp.product_id;

PRODUCT_ID PRODUCT_NAME      ORDER_ID TOTAL_UNITS
---------- --------------- ---------- -----------
       104 Mobile                5100          30
       102 Refrigerator          5101          15
       103 Ipod                  5102          25
       101 Television            5103          10
       100 Camera

SQL> select gkp.product_id, gkp.product_name,gko.order_id,gko.total_units
  2  from gkproduct gkp, gkorder gko
  3  where gkp.product_id(+)=gko.product_id;

PRODUCT_ID PRODUCT_NAME      ORDER_ID TOTAL_UNITS
---------- --------------- ---------- -----------
       101 Television            5103          10
       102 Refrigerator          5101          15
       103 Ipod                  5102          25
       104 Mobile                5100          30
```

NOTE: If the (**+**) operator is used in the left side of the join condition it is equivalent to left outer join. If used on the right side of the join condition it is equivalent to right outer join.

OUTER JOIN :

Outer Join retrieves Either, the matched rows from one table and all rows in the other table Or, all rows in all tables (it doesn't matter whether or not there is a match).

There are three kinds of Outer Join :

LEFT OUTER JOIN or LEFT JOIN

This join returns all the rows from the left table in conjunction with the matching rows from the right table. If there are no columns matching in the right table, it returns NULL values.

RIGHT OUTER JOIN or RIGHT JOIN

This join returns all the rows from the right table in conjunction with the matching rows from the left table. If there are no columns matching in the left table, it returns NULL values.

FULL OUTER JOIN or FULL JOIN

This join combines left outer join and right outer join. It returns row from either table when the conditions are met and returns null value when there is no match.

In other words, OUTER JOIN is based on the fact that : ONLY the matching entries in ONE OF the tables (RIGHT or LEFT) or BOTH of the tables(FULL) SHOULD be listed.

CROSS Join

It is the Cartesian product of the two tables involved. It will return a table with consists of records which combines each row from the first table with each row of the second table.

The result of a CROSS JOIN will not make sense in most of the situations. Moreover, we won't need this at all (or needs the least, to be precise).

```
SQL> select * from gkproduct
  2  CROSS JOIN
  3  gkorder;

PRODUCT_ID PRODUCT_NAME    SUPPLIER_NAME        PRICE   ORDER_ID PRODUCT_ID TOTAL_UNITS CUSTOMER_NAME
---------- --------------- --------------- ---------- ---------- ---------- ----------- -------------
       100 Camera          Nikon               30000       5100        104          30 Infosys
       101 Television      Onida               15000       5100        104          30 Infosys
       102 Refrigerator    videocon            18000       5100        104          30 Infosys
       103 Ipod            Apple               16000       5100        104          30 Infosys
       104 Mobile          Samsung              8000       5100        104          30 Infosys
       100 Camera          Nikon               30000       5101        102          15 GKMV
       101 Television      Onida               15000       5101        102          15 GKMV
       102 Refrigerator    videocon            18000       5101        102          15 GKMV
       103 Ipod            Apple               16000       5101        102          15 GKMV
       104 Mobile          Samsung              8000       5101        102          15 GKMV
       100 Camera          Nikon               30000       5102        103          25 Wipro

PRODUCT_ID PRODUCT_NAME    SUPPLIER_NAME        PRICE   ORDER_ID PRODUCT_ID TOTAL_UNITS CUSTOMER_NAME
---------- --------------- --------------- ---------- ---------- ---------- ----------- -------------
       101 Television      Onida               15000       5102        103          25 Wipro
       102 Refrigerator    videocon            18000       5102        103          25 Wipro
       103 Ipod            Apple               16000       5102        103          25 Wipro
       104 Mobile          Samsung              8000       5102        103          25 Wipro
       100 Camera          Nikon               30000       5103        101          10 TCS
       101 Television      Onida               15000       5103        101          10 TCS
       102 Refrigerator    videocon            18000       5103        101          10 TCS
       103 Ipod            Apple               16000       5103        101          10 TCS
       104 Mobile          Samsung              8000       5103        101          10 TCS

20 rows selected.
```

INNER JOIN: returns rows when there is a match in both tables.

LEFT JOIN: returns all rows from the left table, even if there are no matches in the right table.

RIGHT JOIN: returns all rows from the right table, even if there are no matches in the left table.

FULL JOIN: returns rows when there is a match in one of the tables.

SELF JOIN: is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.

CARTESIAN JOIN: returns the Cartesian product of the sets of records from the two or more joined tables.