Introduction:

What is SQL?

SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in relational database.

SQL is the standard language for Relation Database System. All relational database management systems like MySQL, MS Access, Oracle, Sybase, Informix and SQL Server use SQL as standard database language.

SQL (Structured Query Language) is a language used to access data from the database. The SQL is used for doing various database operations like creating database objects, manipulating data in the database and controlling access to the data.

Why SQL?

- Allows users to access data in relational database management systems.
- Allows users to define the data in database and manipulate that data.
- Allows users to create and drop databases and tables.
- Allows users to create view, stored procedure, functions in a database.
- Allows users to set permissions on tables, procedures, and views.

Note: CEO of Oracle Corporation is "**Larry Ellison**"

History of Oracle: (Different versions of Oracle)

- In 1977, SEL (Software Development Laboratory) … V1
- In 1979, RSI (Relational S/W Incorporation) …. V2
- In 1983, Oracle Corporation → Oracle 3 [Developed Using 'C', which supports simple queries but does not support transactions]
- In 1984, Oracle 4 → Supports Transactions [Commit/Rollback]
- In 1985, Oracle 5 → Client-Server Architecture [Only install DB in Server, so that 'N' no of Clients can connect is known as Client-Server Architecture].
- In 1989, Oracle 6 → PL/SQL
- In 1992, Oracle 7 → Supports DWH [OLAP-Online Analytical Processing]
- In 1997, Oracle 8 → ORDMBS
- In 1999, Oracle 8i → 'I' means Internet & it has inbuilt JVM (JAVA Virtual Machine)
- In 2001, Oracle 9i → with 400 New features, e.g. XML (Xtended Markup Language), RAC (Real Application Clusters) etc which provided high availability & performance.
- In 2003, Oracle 10g → 'g' means grid (group of DB Servers)
- In 2006, Oracle 11g →we can add columns with values etc.

SQL Commands:

The standard SQL commands to interact with relational databases are CREATE, SELECT, INSERT, UPDATE, DELETE and DROP. These commands can be classified into groups based on their nature:

DDL - Data Definition Language:

Command	Description
CREATE	Creates a new table, a view of a table, or other object in database
ALTER	Modifies an existing database object, such as a table.
DROP	Deletes an entire table, a view of a table or other object in the database.

DML - Data Manipulation Language:

Command	Description
INSERT	Creates a record
UPDATE	Modifies records
DELETE	Deletes records

DCL - Data Control Language:

Command	Description
GRANT	Gives a privilege to user
REVOKE	Takes back privileges granted from user

DQL - Data Query Language:

Command	Description
SELECT	Retrieves certain records from one or more tables

What is Database?

1) It is a collection of Inter-Related data. Records the data in HDD (Permanent Memory).
2) Inter-Related data means relation among data values
3) Objective of DB is to record data & save it for future use.

What is RDBMS?

RDBMS stands for **R**elational **D**ata**B**ase **M**anagement **S**ystem. RDBMS is the basis for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.

A Relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as introduced by E. F. Codd.

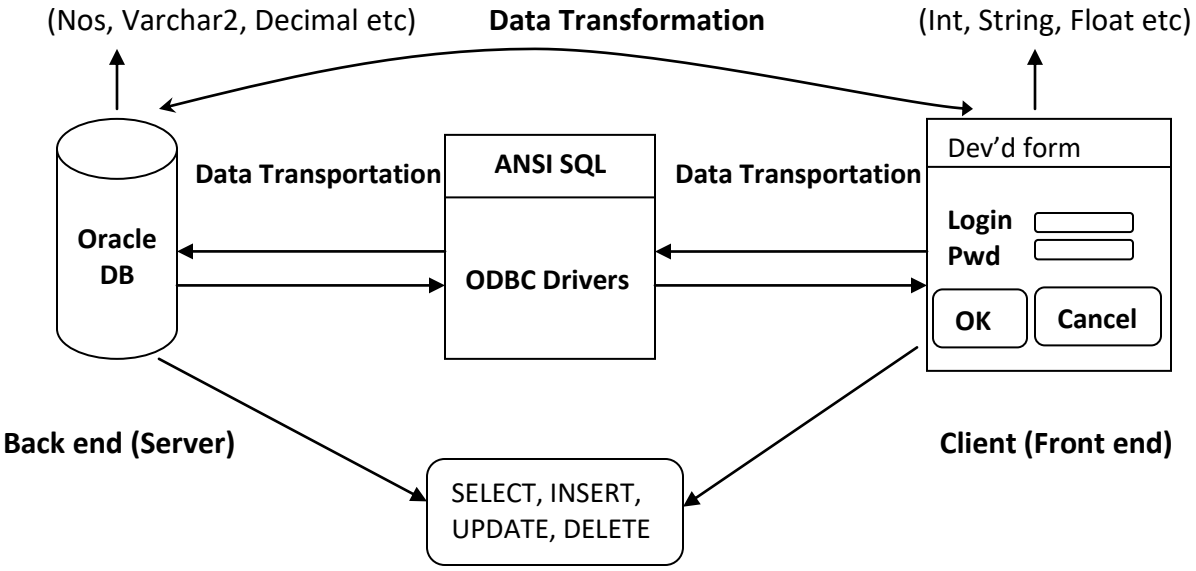
ODBC: (Open DB Connectivity)

1) It is open to all S/W & standardized by ANSI (American National Standard Institute).
2) It acts as an Interface between two different S/W.
3) Main job of ODBC drivers is Data Transformation & Data Transportation.

Data Transformation: It converts one S/W Data Types into another S/W corresponding Data Types.

Data Transportation: It carries instructions from front end (S/W application) & transfers to back end (DB) and vice-versa.

Note: Database is a Client-Server Architecture.



What is table?

The data in RDBMS is stored in database objects called **tables**. The table is a collection of related data entries and it consists of columns (Attributes) and rows (Tuples).

Remember, a table is the most common and simplest form of data storage in a relational database. Following is the example of a CUSTOMERS table:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | SAIF     |  27 | NIBM      |  2000.00 |
|  2 | MANALI   |  25 | YERWADA   |  1500.00 |
|  3 | AMIK     |  23 | JAMMU     |  2000.00 |
|  4 | KAWAL    |  25 | WAKAD     |  6500.00 |
|  5 | VINEETA  |  27 | WAKAD     |  8500.00 |
|  6 | SAM      |  22 | MUMBAI    |  4500.00 |
|  7 | MUFEED   |  24 | PUNE      | 10000.00 |
+----+----------+-----=+-----------+----------+
```

What is field?

Every table is broken up into smaller entities called fields. The fields in the CUSTOMERS table consist of ID, NAME, AGE, ADDRESS and SALARY.

A field is a column in a table that is designed to maintain specific information about every record in the table.

What is record or row?

A record, also called a row of data, is each individual entry that exists in a table. For example there are 7 records in the above CUSTOMERS table. Following is a single row of data or record in the CUSTOMERS table:

```
+----+----------+-----+-----------+----------+
|  1 | SAIF     |  27 | PUNE      |  2000.00 |
+----+----------+-----+-----------+----------+
```

A record is a horizontal entity in a table.

What is column?

A column is a vertical entity in a table that contains all information associated with a specific field in a table. For example, a column in the CUSTOMERS table is ADDRESS, which represents location description and would consist of the following:

```
| ADDRESS   |
+-----------+
| PUNE      |
| NIBM      |
| JAMMU     |
| MUMBAI    |
| BHOPAL    |
| MP        |
| PUNE      |
+----+------+
```

What is NULL value?

A NULL value in a table is a value in a field that appears to be blank, which means a field with a NULL value is a field with no value.

It is very important to understand that a NULL value is different than a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation.

CREATE TABLE:

The CREATE TABLE Statement is used to Create Tables and Store Data; we can Create Table in 3 different ways.

Integrity Constraints like Primary Key, Unique Key, and Foreign Key can be defined for the Columns while creating the Table. The Integrity Constraints can be defined at Column level or Table level.

1) Syntax:

```
CREATE TABLE <Table_Name>
(
Column_Name1 Data_Type (Size),
Column_Name2 Data_Type (Size),
Column_Name3 Data_Type (Size),
…
…
Column_NameN Data_Type (Size)
);
```

E.g.

```
CREATE TABLE Student
(
RollNo Number ,
FName Varchar2 (15),
LName Varchar2 (15),
Class Number (10),
DOB Date,
Gender Varchar2 (15),
Location Varchar2 (15),
PhoneNos Number (15)
);
```

Note: If you miss to assign a PK while creating a TABLE then use below Query:

```
ALTER TABLE <Table_Name>
MODIFY Column_Name Data_Type (Size) PRIMARY KEY;
```

E.g.

```
ALTER TABLE Student
MODIFY RollNo Number (10) PRIMARY KEY;
```

2) Syntax:

CREATE TABLE <Table_Name>
AS SELECT * FROM <Table_Name>;

E.g.

CREATE TABLE Student_Temp
AS SELECT * FROM Student;

Note: In the above statement, **Student_Temp** Table is created with the same number of Columns and Data_Type as **Student** Table but no CONSTRAINTS are carried over.

3) Syntax:

CREATE TABLE <Table_Name>
AS SELECT **<Column_Name1>,<Column_Name2>…**FROM <Table_Name>;

E.g.

CREATE TABLE Student_TEMP1
AS SELECT **RollNo, FName, LName** FROM Student;

Note: In the above statement, **Student_TEMP1** Table is created with the Specified number of Columns and Data_Type as **Student** Table but no CONSTRAINTS are carried over.
Also, while writing Column names you should not include Column names in parenthesis.

ALTER TABLE:

The ALTER TABLE command is used to modify the Definition/Structure of a Table by modifying the definition of its columns. The ALTER command is used to perform the following functions:

1) Add, Modify, and Drop Columns
2) Add and Drop Constraints
3) Enable and Disable Constraints

a) Syntax to **ADD** a Column:

ALTER TABLE <Table_Name>
ADD <Column_Name> Data_Type (Size);

E.g.

ALTER TABLE Student
ADD Percentage Varchar2 (10);

b) Syntax to **MODIFY** a Column:

ALTER TABLE <Table_Name>
MODIFY <Column_Name> Data_Type (Size);

E.g.

ALTER TABLE Student
MODIFY PhoneNos Varchar2 (10);

ALTER TABLE Student
MODIFY PhoneNos Number (15);

Note:
a) In the above Statement MODIFY means, you can change the Data_Type & Size of the Column
b) To change the Data_Type, Column Data must be Empty but you can Increase/Decrease size.

If you want to change the Column_Name then you can use below syntax:

Syntax:
ALTER TABLE <Table_Name>
RENAME COLUMN <Old_Column_Name> TO <New_Column_Name>;

E.g.
ALTER TABLE TEST
RENAME COLUMN EMPNO TO SRNO;

c) Syntax to **DROP** a Column:

ALTER TABLE <Table_Name>
DROP COLUMN <Column_Name>;

E.g.

ALTER TABLE Student
DROP COLUMN Percentage;

d) Syntax to **ADD a Constraint**: [Existing column from a table]

ALTER TABLE <Table_Name>
ADD PRIMARY KEY <Column_Name>;

ALTER TABLE <Table_Name>
MODIFY <Column_Name> PRIMARY KEY;

ALTER TABLE <Table_Name>
MODIFY <Column_Name> Data_Type (Size) PRIMARY KEY;

Note: In the above queries, PRIMARY KEY can be added & at the same time Data_Type Size can be Increased or Decreased. Also, a Table can have only **ONE** PRIMARY KEY.

E.g.

ALTER TABLE Student
ADD PRIMARY KEY (RollNo);

```
ALTER TABLE Student
MODIFY RollNo PRIMARY KEY;
```

```
ALTER TABLE Student
MODIFY RollNo Number (10) PRIMARY KEY;
```

e) Syntax to **DROP a Constraint**:

```
ALTER TABLE <Table_Name>
DROP <Constraint_Name>;
```

E.g.

```
ALTER TABLE Student
DROP PRIMARY KEY;
```

SQL RENAME:

The SQL RENAME Command is used to change the name of the Table or a Database Object.
If you change the object's name any reference to the Old Name will be affected. You have to manually change the Old Name to the new name in every reference.

Syntax to **RENAME** a Table:

```
RENAME <Old_Table_Name> To <New_Table_Name>;
```

E.g.

```
RENAME Student TO TEMP;
```

Note: New Table Name should not be an existing Table Name in Database.

DROP TABLE:

The DROP TABLE Statement is used to DELETE the Rows in the Table and the Table Structure is removed from the Database.

Syntax:

```
DROP TABLE <Table_Name>;
```

E.g.

```
DROP TABLE Student;
```

Note: Once a Table is dropped we cannot get it back through ROLLBACK because DROP is a DDL Command.

TRUNCATE TABLE:

What if we only want to delete the data inside the table, and not the table itself? And want to free the space then use TRUNCATE TABLE statement:

Syntax:

TRUNCATE TABLE <Table_Name>;

E.g.
TRUNCATE TABLE EMP;

Note: Once a Table is truncated we cannot get it back through ROLLBACK because Truncate is a DDL Command.

DELETE Statement:

The DELETE Statement is used to delete rows in a Table.

Syntax:

DELETE FROM <Table_Name>
WHERE Some_Column=Some_Value;

DELETE FROM EMP
WHERE Ename='SMITH';

DELETE FROM EMP
WHERE EmpNo=7369 or Comm=300;

Note: The WHERE Clause specifies which record or records that should be Deleted. If you omit the WHERE Clause, all records will be deleted!

Delete All Data:

It is possible to delete all rows in a Table without deleting the table. This means that the Table Structure, Attributes, and Indexes will be intact.

DELETE FROM <Table_Name>;

E.g.

DELETE FROM EMP;

***** Difference between DELETE, TRUNCATE and DROP Statements: *****

DELETE Statement:

This Command Deletes only the rows from the Table based on the Condition given in the WHERE Clause or Deletes all the rows from the Table if NO Condition is specified. But it does not FREE the space from the Table.

TRUNCATE Statement:

This Command is used to Delete all the rows from the Table and FREE the space from the Table.

DROP Statement:

This Command is used to remove an Object from the Database. If you DROP a Table, all the rows in the Table are Deleted and the Table Structure is removed from the Database. Once a Table is dropped we cannot get it back through ROLLBACK, Also be careful while using RENAME command. When a Table is dropped all the references to the Table will not be valid.

SQL INSERT INTO Statement:

The INSERT INTO Statement is used to insert new records in a Table. It is possible to write the INSERT INTO Statement in 3 forms.

1) The First Form does not specify the Column Names where the Data will be inserted, we can directly enter their values BUT all the Column Values should be entered otherwise it will throw an error.

INSERT INTO <Table_Name>
VALUES (Value1, Value2, Value3, Value4, Value5);

E.g.

INSERT INTO Student
Values (101,'Saif','Shaikh',10,'14-Apr-85','NIBM',12345);

2) The Second Form specifies both the Column Names and the Values to be inserted:

INSERT INTO <Table_Name> (Column1, Column2, Column3, Column4, Column5)
VALUES (Value1, Value2, Value3, Value4, Value5);

E.g.

INSERT INTO Student (RollNo, FName, LName, Class, DOB, Gender, Location, PhoneNos)
Values (102,'Saif','Khan', 10,'10-Jun-90','Male','Hadapsar', 54264);

Note: You can also specify Fewer Columns and their Values instead of ALL Columns.

E.g.

INSERT INTO Student (RollNo, FName, LName, Gender)
Values (103,'Miraj','Khan','Female');

3) If you want to enter Multiple Columns Values without writing column name always.

INSERT INTO <Table_Name>
Values (&Column1, &Column2, &Column3, &Column4, &Column5);

E.g.

INSERT INTO Student
Values (&RollNo, &FName, &LName, &Class, &DOB, &Location);

Note: Once you Enter all the Column values, Enter **/** to Re-rerun and enter the values again. You cannot specify fewer columns; need to specify all the columns from the table.

INSERTING Data to a Table through a SELECT Statement can be done in 2 ways:

1) If you are inserting data to all the columns, the Insert Statement can be written as:

Syntax:

INSERT INTO <Table_Name>
SELECT * FROM <Table_Name>;

E.g.

INSERT INTO Student_Temp
SELECT * FROM Student;

2) If you are inserting data to specified columns, the Insert Statement can be written as:

Syntax:

INSERT INTO <Table_Name> [(Column1, Column2,...ColumnN)]
SELECT Column1, Column2,...ColumnN FROM <Table_Name> [WHERE condition];

E.g.

INSERT INTO Student_TEMP1 (RollNo, FName, LName, Class)
SELECT RollNo, FName, LName, Class FROM Student;

INSERT INTO Student_TEMP1 (RollNo, FName, LName)
SELECT RollNo, FName, LName FROM student
WHERE Gender='Male';

Note:
1) Column Names should be in the Same Order & Data_Type should also be the same.
2) When adding a new row, you should ensure the Data_Type of the Value and the Column matches.
3) You follow the Integrity Constraints, if any, defined for the Table.

SQL SELECT Statements

The SQL SELECT Statement is used to Query or Retrieve Data from a Table in the Database. A Query may retrieve information from Specified Columns or from all of the Columns in the Table. To create a simple SQL SELECT Statement, you must specify the column(s) name and the Table Name.

Syntax:
SELECT Column_Name(s) FROM <Table_Name>
[WHERE clause]
[GROUP BY clause]
[HAVING clause]
[ORDER BY clause];

E.g.

SELECT * FROM Student;

SELECT FName, LName FROM Student;

SELECT FName||'-'||LName AS Name
FROM Student;

The **DISTINCT** Keyword can be used to return only distinct (different) values.

E.g.

SELECT DISTINCT Column_Name(s)
FROM <Table_Name>;

SELECT DISTINCT FName||'-'||LName AS Name
FROM Student;

WHERE Clause:

WHERE Clause is used to extract only those records that fulfill a specified criteria.

Syntax:
SELECT Column_Name(s)
FROM <Table_Name>
WHERE Column_Name=Operator Value;

E.g.
SELECT FName, LName
FROM Student
WHERE Class=10;

Operators in the WHERE Clause:

The following operators can be used in the WHERE clause:

Operator	Description
=	Equal
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

E.g.

```
SELECT * FROM Student
WHERE Class=10;

SELECT * FROM Student
WHERE Class<>10;

SELECT * FROM Student
WHERE Class>4;

SELECT * FROM Student
WHERE RollNo>4 or Class>4;

SELECT * FROM Student
WHERE RollNo<2;

SELECT * FROM Student
WHERE DOB>='01-Jan-90';

SELECT * FROM Student
WHERE DOB<='01-Jan-90';

SELECT * FROM Student
WHERE RollNo BETWEEN 2 AND 4;

SELECT * FROM Student
WHERE Location LIKE 'NIBM';

SELECT * FROM Student
WHERE Class IN (10, 12);
```

AND/OR Operators:

The AND Operator displays a record if both the First Condition AND the Second Condition is TRUE.
The OR Operator displays a record if either the First Condition OR the Second Condition is TRUE.

AND Operator:

The following SQL statement selects all names of the students from the Student Table WHERE FName=’Saif’ AND LName=’Shaikh’.

E.g.

```
SELECT * FROM Student
WHERE FName='Saif' AND LName='Shaikh';
```

Note: Data inside the Table is Case Sensitive.

OR Operator:

The following SQL statement selects all names of the students from the Student Table WHERE FName=’Saif’ OR LName=’Khan’.

E.g.

```
SELECT * FROM Student
WHERE FName=’Saif’ OR LName=’Khan’;
```

Combining AND/OR Operators:

You can also combine AND/OR Operators. (Using parenthesis to form complex expressions)

The following SQL Statement selects all names of student from the Student Table WHERE Class=10 AND FName=’Saif’ OR LName=’Chormare’.

E.g.

```
SELECT * FROM Student
WHERE Class=10 AND (FName='Saif' OR LName='Shaikh');

SELECT * FROM student
WHERE Class=10 OR (FName='Saif' AND LName='Khan');
```

The SQL ORDER BY Keyword:

The ORDER BY Keyword is used to Sort the Result-Set by One or More Columns.
The ORDER BY Keyword Sorts the records in Ascending order by default. To sort the records in a Descending order, you can use the DESC keyword.

Syntax:

SELECT Column_Name(s)
FROM <Table_Name>
ORDER BY Column_Name(s) **ASC|DESC**;

By Default, it is ASCENDING.

E.g.

SELECT FName FROM Student
ORDER BY FName;

SELECT FName FROM Student
ORDER BY FName DESC;

Note: You can use Position Number also. For e.g. 1 or 2 or 3 etc

E.g.

SELECT FName, LName, Location
FROM Student
ORDER BY 2 DESC;

The SQL UPDATE Statement:

The UPDATE Statement is used to Update/Modify existing records in a Table.

Syntax:

UPDATE <Table_Name>
SET Column1=Value1, Column2=Value2,…ColumnN=ValueN
WHERE Some_Column=Some_Value;

E.g.

UPDATE Student
SET Class=5, DOB='10-Jan-97'
WHERE RollNo=106;

Note: The WHERE Clause specifies which record or records that should be updated. If you Omit the WHERE clause, all records will be updated!

E.g.

UPDATE Student_Temp
SET Gender='N/A';

The SQL DELETE Statement:

The DELETE Statement is used to delete rows in a Table.

Syntax:
DELETE FROM <Table_Name>
WHERE Some_Column=Some_Value;

DELETE FROM Student_Temp
WHERE RollNo=105;

Note: The WHERE Clause specifies which record or records that should be Deleted. If you omit the WHERE Clause, all records will be deleted!

Delete All Data:

It is possible to delete all rows in a Table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

Syntax:
DELETE FROM <Table_Name>;

E.g. DELETE FROM Student_Temp;

SQL Constraints:

SQL constraints are used to specify rules for the data in a table. If there is any violation between the constraint and the data action, the action is aborted by the constraint.

Constraints can be specified when the table is created (inside the CREATE TABLE statement) or after the table is created (inside the ALTER TABLE statement).

Constraints could be column level or table level. Column level constraints are applied only to one column whereas table level constraints are applied to the whole table.

In SQL, we have the following constraints:

- **NOT NULL** - Indicates that a column cannot store NULL value
- **UNIQUE** - Ensures that each row for a column must have a unique value
- **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Ensures that a column (or combination of two or more columns) have an unique identity which helps to find a particular record in a table more easily and quickly
- **FOREIGN KEY** - Ensure the referential integrity of the data in one table to match values in another table
- **CHECK** - Ensures that the value in a column meets a specific condition
- **DEFAULT** - Specifies a default value when specified none for this column
- **INDEX:** Use to create and retrieve data from the database very quickly.

1) SQL NOT NULL Constraint:

By default, a table column can hold NULL values. The NOT NULL constraint enforces a column to NOT accept NULL values.

The NOT NULL constraint enforces a field to always contain a value. This means that you cannot insert a new record, or update a record without adding a value to this field.

The following SQL enforces the "RollNo" Column to NOT accept NULL values:

```
CREATE TABLE Student1
(
RollNo Number (10) NOT NULL,
FName Varchar2 (15),
LName Varchar2 (15),
Location Varchar2 (20)
);
```

E.g.

```
INSERT INTO Student1
VALUES (101, 'Saif', 'Shaikh', 'Kondhwa');

INSERT INTO Student1 (RollNo, FName, LName, Location)
Values ('','Saif', 'Shaikh', 'Kondhwa');
```

ERROR at line 2:
ORA-01400: cannot insert NULL into ("SAIF"."STUDENT1"."ROLLNO")

Note: NOT NULL Constraint can be defined only at a Column level.
To DROP NOT NULL Constraint:
To Drop a NOT NULL Constraint, use the following SQL:

Syntax:

ALTER TABLE <Table_Name>
DROP CONSTRAINT <Constraint_Name>;

To find System Constraint:
SQL→SELECT Table_Name, Constraint_Name FROM User_Constraints;

E.g.

ALTER TABLE Student1
DROP Constraint SYS_C005021;

2) SQL UNIQUE Constraint:

The UNIQUE Constraint uniquely identifies each record in a database table.

The UNIQUE and PRIMARY KEY Constraints both provide a guarantee for Uniqueness for a column or set of columns.

A PRIMARY KEY Constraint automatically has a UNIQUE Constraint defined on it.

Note: You can have many UNIQUE Constraints per table, but only one PRIMARY KEY Constraint per table is allowed.

CREATE TABLE Student2
(
RollNo Number (10) UNIQUE,
FName Varchar2 (15),
LName Varchar2 (15),
Location Varchar2 (20)
);

Try Entering below Query twice:

E.g.

INSERT INTO Student2 (RollNo, FName, LName, Location)
Values (101,'Saif', 'Shaikh', 'Kondhwa');

ERROR at line 1:
ORA-00001: unique constraint (SAIF.SYS_C004929) violated

To find System Constraint with Constraint Type:
SQL→SELECT Table_Name, Constraint_Type, Constraint_Name FROM User_Constraints;

To allow naming of a UNIQUE constraint, and for defining a UNIQUE constraint on multiple columns, use the following

SQL Syntax:

```
CREATE TABLE Student2A
(
RollNo Number (10),
FName Varchar2 (15),
LName Varchar2 (15),
Location Varchar2 (20),
CONSTRAINT U_Student2A_FName_LName UNIQUE (FName, LName)
);
```

Try Entering below Query twice:

E.g.

```
INSERT INTO Student2A (RollNo, FName, LName, Location)
Values (101,'Saif', 'Shaikh', 'Kondhwa');
```

ERROR at line 1:
ORA-00001: unique constraint (SAIF.U_STUDENT2A_FNAME_LNAME) violated

To DROP a UNIQUE Constraint:

To Drop a UNIQUE Constraint, use the following SQL:

Syntax:

```
ALTER TABLE <Table_Name>
DROP CONSTRAINT <Constraint_Name>;
```

E.g.

```
ALTER TABLE Student2
DROP CONSTRAINT SYS_C005059;
```

E.g.

```
ALTER TABLE Student2A
DROP CONSTRAINT U_Student2A_FName_LName;
```

Note: The above statement will drop the Unique Constraint 'U_Student2A_FName_LName' from the 'Student_2A' Table.

SQL UNIQUE Constraint on ALTER TABLE:

To create a UNIQUE constraint on the "RollNo" Column when the table is already created, use below SQL:

SQL Syntax:

ALTER TABLE <Table_Name>
ADD <Constraint_Type> (Column_Name);

ALTER TABLE Student2
ADD Unique (RollNo);

OR

ALTER TABLE Student2 MODIFY RollNo Number (5) UNIQUE;
ALTER TABLE Student2 MODIFY RollNo Unique;

To allow naming of a UNIQUE Constraint, and for defining a UNIQUE constraint on multiple columns, use the following

SQL Syntax:

ALTER TABLE <Table_Name>
ADD CONSTRAINT <Constraint_Name> CONSTRAINT_TYPE (Column_Name);

ALTER TABLE Student2A
ADD CONSTRAINT U_Student2A_FName_Location UNIQUE (FName, Location);

Disable a UNIQUE CONSTRAINT:

If we do not wish to DELETE the Unique Constraint as we may need the same in future but for some time we want the unique constraint not to function, then we can DISABLE the Unique Constraint.

Syntax:

ALTER TABLE <Table_Name>
DISABLE CONSTRAINT <Constraint_Name>;

To find Constraint Enabled or Disabled:
SQL→Select Table_Name, Constraint_Type, Constraint_Name, Status, Generated from User_Constraints;

E.g.
ALTER TABLE Student2A
DISABLE CONSTRAINT U_Student2A_FName_Location;

Here in the above ALTER Statement we have DISABLED the UNIQUE CONSTRAINT '<Constraint_Name>' on the 'Student2A' Table.

Enabling a UNIQUE CONSTRAINT:

We can enable a unique constraint that has been disabled earlier, the syntax for enabling a unique constraint in Oracle SQL / PLSQL is:

Syntax:

```
ALTER TABLE <Table_Name>
ENABLE CONSTRAINT <Constraint_Name>;
```

E.g.

```
ALTER TABLE Student2A
ENABLE CONSTRAINT U_Student2A_FName_Location;
```

Here in the above ALTER Statement we have ENABLED the UNIQUE CONSTRAINT '<Constraint_Name>' on the 'Student2A' Table.

3) SQL PRIMARY KEY Constraint:

The PRIMARY KEY Constraint Uniquely identifies each record in a database table.

PRIMARY KEY must contain unique values. A Primary Key column cannot contain NULL values. Each table can have only **ONE PRIMARY KEY**.

Syntax:

```
CREATE TABLE Student3
(
RollNo Number (10) PRIMARY KEY,
FName Varchar2 (15),
LName Varchar2 (15),
Location Varchar2 (10)
);
```

To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY Constraint on multiple columns, use the following SQL syntax:

```
CREATE TABLE Student3A
(
RollNo Number (10),
FName Varchar2 (15),
LName Varchar2 (15),
Location Varchar2 (10),
CONSTRAINT PK_Student3A_FName_LName PRIMARY KEY (FName, LName)
);
```

Note: In the example above there is only ONE PRIMARY KEY (PK_Student3A_FName_LName). However, the value of the PK_Student3A_FName_LName is made up of two columns (FName and LName).

To DROP a PRIMARY KEY Constraint:

To Drop a Primary Key Constraint, use the following SQL:

Syntax:

ALTER TABLE <Table_Name>
DROP CONSTRAINT <Constraint_Name>;

E.g.

ALTER TABLE Student3
DROP CONSTRAINT SYS_C005063;

E.g.

ALTER TABLE Student3A
DROP CONSTRAINT PK_Student3A_FName_LName;

Note: The above statement will drop the Primary Key Constraint 'PK_Student3A_FName_LName' from the 'Student_3A' Table.

SQL PRIMARY KEY Constraint on ALTER TABLE:

To create a PRIMARY KEY Constraint on the "RollNo" column when the table is already created, use the following SQL:

Syntax:

ALTER TABLE <Table_Name>
ADD <Constraint_Type> (Column_Name);

E.g.

ALTER TABLE Student3
ADD PRIMARY KEY (RollNo);

OR

ALTER TABLE Student3 MODIFY RollNo Number (10) Primary Key;
ALTER TABLE Student3 MODIFY RollNo Primary Key;

To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

ALTER TABLE <Table_Name>
ADD CONSTRAINT <Constraint_Name> CONSTRAINT_TYPE (Column_Name);

```
ALTER TABLE Student3A
ADD CONSTRAINT PK_Student3A_FName_LName PRIMARY KEY (FName, LName);
```

Note: If you use the ALTER TABLE statement to add a primary key, the primary key column(s) must already have been declared to not contain NULL values (when the table was first created).

To find Selected Tables from a User:

SQL→ SELECT Table_Name FROM User_Tables;

SQL→SELECT Table_Name FROM User_Tables WHERE Table_Name LIKE 'STU%';

To find Selected Tables which have Constraints:

SQL→

```
SELECT Table_Name, Constraint_Name, Constraint_Type, Status, Generated
FROM User_Constraints
WHERE Table_Name LIKE 'STU%';
```

4) SQL FOREIGN KEY Constraint

A FOREIGN KEY in one table points to a PRIMARY KEY in another table. The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

The FOREIGN KEY constraint also prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the table it points to.

TO find Columns on which Constraints are declared:

SQL→

```
SELECT Column_Name, Constraint_Name FROM User_Cons_Columns;
SELECT Column_Name, Constraint_Name, Table_Name
FROM User_Cons_Columns WHERE Table_Name='STUDENT3';
```

Syntax:

```
CREATE TABLE <Table_Name>
(
Column1 Data_Type (size) REFERENCES <Parent_Table_Name> (Column_Name), [Column_Level]
Column2 Data_Type (size),
Column3 Data_Type (size),
CONSTRAINT <Constraint_Name> FOREIGN KEY (Column_Name)
REFERENCES <Parent_Table_Name> <Column_Name> [Table Level]
);
```

E.g.

```
CREATE TABLE Student3FK
(
Class Number (5),
RollNo Number (10) REFERENCES Student3 (RollNo),
```

```
Subject Varchar2 (15),
Fees Number (10, 2)
);

CREATE TABLE Student3FK
(
Class Number (5),
RollNo Number (10),
Subject Varchar2 (15),
Fees Number (10, 2),
FOREIGN KEY (RollNo) REFERENCES Student3 (RollNo)
);

CREATE TABLE Student3AFK
(
Class Number (5),
FName Varchar2 (15),
LName Varchar2 (15),
Subject Varchar2 (15),
Fees Number (10, 2),
CONSTRAINT FK_Student3AFK_FName_LName FOREIGN KEY (FName, LName)
REFERENCES Student3A (FName, LName)
);
```

To DROP a FOREIGN KEY Constraint:

Syntax:

```
ALTER TABLE <Table_Name>
DROP CONSTRAINT <Constraint_Name>;
```

E.g.

```
ALTER TABLE Student3FK
DROP Constraint SYS_C005028;

ALTER TABLE Student3AFK
DROP CONSTRAINT FK_Student3AFK_FName_LName;
```

SQL FOREIGN KEY Constraint on ALTER TABLE:

To create a FOREIGN KEY Constraint on the "RollNo" Column when the "Student3" table is already created, use the following SQL:

Syntax:

```
ALTER TABLE Student3FK
ADD FOREIGN KEY (RollNo) REFERENCES Student3 (RollNo);
```

To allow naming of a FOREIGN KEY Constraint, and for defining a FOREIGN KEY Constraint on multiple Columns, use the following SQL Syntax:

Syntax:

```
ALTER TABLE Student3AFK
ADD CONSTRAINT FK_Student3AFK_FName_LName
FOREIGN KEY (FName, LName) REFERENCES Student3A (FName, LName);
```

5) SQL CHECK Constraint:

The CHECK constraint is used to limit the value range that can be placed in a column. If you define a CHECK constraint on a single column it allows only certain values for this column.

If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

```
CREATE TABLE Student4
(
Class Number (5),
RollNo Number (10),
Subject Varchar2 (15),
Fees Number (10, 2) CHECK (Fees>500)
);
```

E.g.

```
INSERT INTO Student4 (Class, RollNo, Subject, Fees)
Values (5, 105, 'Eng', 550);

INSERT INTO Student4 (Class, RollNo, Subject, Fees)
Values (5, 105, 'Eng', 50);
```

ERROR at line 1:
ORA-02290: check constraint (SAIF.SYS_C004958) violated

To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:

```
CREATE TABLE Student4A
(
Class Number (5),
RollNo Number (10),
Subject Varchar2 (15),
Fees Number (10),
CONSTRAINT C_Student4A_Fees_Subject CHECK (Fees>500 AND Subject='English')
);
```

E.g.

```
INSERT INTO Student4A (Class, RollNo, Subject, Fees)
Values (5, 10, 'English', 650);

INSERT INTO Student4A (Class, RollNo, Subject, Fees)
Values (5, 10, 'Maths', 350);

ERROR at line 1:
ORA-02290: check constraint (SAIF.C_STUDENT4A_FEES_SUBJECT) violated
```

Note: You can use **AND/OR** Operators in CHECK Constraint.

```
CREATE TABLE Student4B
(
Class Number (5),
RollNo Number (10),
Subject Varchar2 (15),
Fees Number (10),
CONSTRAINT C_Student4B_Fees_Subject CHECK (Fees>500 OR Subject='English')
);
```

E.g.

```
INSERT INTO Student4B (Class, RollNo, Subject, Fees)
Values (5, 10, 'English', 50);

INSERT INTO Student4B (Class, RollNo, Subject, Fees)
Values (5, 10, 'Maths', 650);

INSERT INTO Student4B (Class, RollNo, Subject, Fees)
Values (5, 10, 'Science', 400);
```

ERROR at line 1:

```
ORA-02290: check constraint (SAIF.C_STUDENT4B_FEES_SUBJECT) violated
```

To DROP a CHECK Constraint:

To Drop a CHECK Constraint, use the following SQL.

Syntax:

ALTER TABLE <Table_Name>
DROP CONSTRAINT <Constraint_Name>;

E.g.

ALTER TABLE Student4
DROP Constraint SYS_C005030;

ALTER TABLE Student4A
DROP CONSTRAINT C_Student4A_Fees_Subject;

ALTER TABLE Student4B
DROP CONSTRAINT C_Student4B_Fees_Subject;

SQL CHECK Constraint on ALTER TABLE:

Syntax:

ALTER TABLE <Table_Name>
ADD <Constraint_Type> (Column_Name);

ALTER TABLE Student4
ADD CHECK (RollNo>5);

OR

ALTER TABLE Student4 MODIFY Class Number (10) CHECK (Class>5);

To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:

Syntax:

ALTER TABLE <Table_Name>
ADD CONSTRAINT <Constraint_Name> CONSTRAINT_TYPE (Column_Name);

ALTER TABLE Student4A
ADD CONSTRAINT C_Student4A_Subject_Fees CHECK (Subject='Hindi' AND Fees>500);

ALTER TABLE Student4B
ADD CONSTRAINT C_Student4B_Subject_Fees CHECK (Subject='Hindi' OR Fees>500);

6) SQL DEFAULT Constraint:

The DEFAULT constraint is used to insert a default value into a column.

The default value will be added to all new records, if no other value is specified.

The following SQL creates a DEFAULT constraint on the "Class" column when the "Student5" Table is created:

```
CREATE TABLE Student5
(
RollNo Number (10),
Subject Varchar2 (15),
Fees Number (10),
Class Number (5) DEFAULT 5,
DOJ Date Default SYSDATE
);

INSERT INTO Student5 (RollNo, Subject, Fees)
Values (105, 'Sci', 5000);
```

To DROP a DEFAULT Constraint:

Note: To DROP a Default constraint you need to make is as NOT NULL first, then apply DROP Command.

```
ALTER TABLE Student5 MODIFY Class NOT NULL;
```

Syntax:

```
ALTER TABLE <Table_Name>
DROP CONSTRAINT <Constraint_Name>;
```

E.g.

```
ALTER TABLE Student5
DROP Constraint SYS_C005031;
```

SQL DEFAULT Constraint on ALTER TABLE:

To create a DEFAULT constraint on the "Subject" column when the table is already created, use the following SQL:

E.g.

```
ALTER TABLE Student5
MODIFY Subject DEFAULT 'English';
```

SQL NULL Values:

If a column in a table is optional, we can insert a new record or update an existing record without adding a value to this column. This means that the field will be saved with a NULL value.

By default, a table column can hold NULL values. NULL values represent missing unknown data. NULL values are treated differently from other values. NULL is used as a placeholder for unknown or inapplicable values.

Note: It is not possible to compare NULL and 0; they are not equivalent.

It is not possible to test for NULL values with comparison operators, such as =, <, or <>. We will have to use the IS NULL and IS NOT NULL operators instead.

SQL IS NULL:

Syntax:
SELECT * FROM <Table_Name> WHERE <Column_Name> IS NULL

E.g.
SELECT * FROM Student_Temp WHERE Location IS NULL;

Note: Always use IS NULL to look for NULL values.

SQL IS NOT NULL:

Syntax:
SELECT * FROM <Table_Name> WHERE <Column_Name> IS NOT NULL

E.g.
SELECT * FROM Student_Temp WHERE Location IS NOT NULL;

NVL: If we are querying on a column and the value of that placeholder is NULL, then the result would be NULL.

E.g.
SELECT E.*, Sal + Comm as Salary
FROM EMP E;

NVL () function returns a zero if the value is NULL. So the above query would be written as below:
E.g.
SELECT E.*, Sal +NVL (Comm, 0) as Salary
FROM EMP E;

Also, we can use the COALESCE () function, like this:

SELECT E.*, Sal + COALESCE (Comm, 0) as Salary
FROM EMP E;

SQL Functions:

SQL has many built-in functions for performing calculations on data.

SQL Aggregate Functions

SQL aggregate functions return a single value, calculated from values in a column.

Useful aggregate functions:

- AVG() - Returns the average value
- COUNT() - Returns the number of rows
- MAX() - Returns the largest value
- MIN() - Returns the smallest value
- SUM() - Returns the sum

SQL Scalar functions

SQL scalar functions return a single value, based on the input value.

Useful scalar functions:

- UCASE() - Converts a field to upper case
- LCASE() - Converts a field to lower case
- SUBSTR() - Extract characters from a text field
- LEN() - Returns the length of a text field
- ROUND() - Rounds a numeric field to the number of decimals specified

1) The AVG () Function:

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

Syntax:

```
SELECT AVG (Column_Name)
FROM <Table_Name>;
```

E.g.

```
SELECT AVG (Sal)
FROM EMP;

SELECT DeptNo, AVG (Sal) FROM EMP
GROUP BY DeptNo;

SELECT DeptNo, AVG (Sal) AS AvgSal FROM EMP
GROUP BY DeptNo
ORDER BY DeptNo;

SELECT EMPNo, Ename, Job, Sal, DeptNo FROM EMP
WHERE Sal > (SELECT AVG (Sal) FROM EMP);
```

2) The COUNT () Function:

The COUNT () function returns the number of rows that matches a specified criteria.

1) COUNT (Column_Name)
2) COUNT (*)
3) COUNT (DISTINCT Column_Name)

1) Syntax:
SELECT COUNT (Column_Name)
FROM <Table_Name>;

SQL→ SELECT COUNT (Ename) FROM EMP;

SQL→SELECT Job, COUNT (Job) FROM EMP
WHERE Job='CLERK'
GROUP BY Job;

SQL→ SELECT COUNT (Comm) FROM EMP;

Note: The COUNT (Column_Name) function returns the number of values (NULL values will not be counted) of the specified column.

2) Syntax:
SELECT COUNT (*)
FROM <Table_Name>;

SQL→ SELECT COUNT (*) FROM EMP;

Note: The COUNT (*) function returns the total number of records in a table, counts NULL values also.

3) Syntax:
SELECT COUNT (DISTINCT Column_Name)
FROM <Table_Name>;

SQL→ SELECT COUNT (DISTINCT Job) FROM EMP;

SQL→ SELECT COUNT (DISTINCT (Ename)) FROM EMP;

SQL→ SELECT Ename, COUNT (DISTINCT (Ename)) FROM EMP GROUP BY Ename;
SQL→ SELECT Job, COUNT (DISTINCT (Job)) FROM EMP GROUP BY Job;

Note: The COUNT (DISTINCT column_name) function returns the number of distinct values of the specified column.

3) The MAX () Function:

The MAX () function returns the largest value of the selected column.

Syntax:
SELECT MAX (Column_Name)
FROM <Table_Name>;

SQL→ SELECT MAX (Sal) FROM EMP;

SQL→ SELECT DeptNo, Max (Sal) FROM EMP
GROUP BY DeptNo
ORDER BY DeptNo;

4) The MIN () Function:

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

Syntax:
SELECT MIN (Column_Name) FROM <Table_Name>;

SQL→ SELECT MIN (Sal) FROM EMP;

SQL→ SELECT DeptNo, Min (Sal) FROM EMP
GROUP BY DeptNo
ORDER BY DeptNo;

5) The SUM () Function

The SUM () function returns the total sum of a numeric column.

Syntax:
SELECT SUM (Column_Name) FROM <Table_Name>;

SQL→ SELECT SUM (Sal) FROM EMP;

SQL→ SELECT DeptNo, Sum (Sal) FROM EMP
GROUP BY DeptNo
ORDER BY DeptNo;

6) The UPPER () Function:

The UPPER () function converts the value of a field to Upper-Case.

Syntax:
SELECT UPPER (Column_Name) FROM <Table_Name>;

SQL→ SELECT UPPER (Ename) FROM EMP;

7) The LOWER () Function:

The LOWER () function converts the value of a field to Lower-Case.

Syntax:
SELECT LOWER (Column_Name) FROM <Table_Name>;

SQL→ SELECT LOWER (Ename) FROM EMP;

8) The INITCAP () Function:

The INITCAP () function converts the value of a field to Initial-Case.

Syntax:
SELECT INITCAP (Column_Name) FROM <Table_Name>;

SQL→ SELECT INITCAP (Ename) FROM EMP;

9) The SUBSTR () Function:

The SUBSTR () function is used to extract characters from a text field.

Syntax:
SELECT SUBSTR (Column_Name, Start Position, Length)
FROM <Table_Name>;

Parameter	Description
column_name	Required. The field to extract characters from
start	Required. Specifies the starting position (starts at 1)
length	Optional. The number of characters to return. If omitted, the MID() function returns the rest of the text

SQL→ SELECT Substr (Ename, 1, 3) FROM EMP;

10) The LENGTH () Function:

The LENGTH () function returns the length of the value in a text field.

Syntax:
SELECT LENGTH (Column_Name)
FROM <Table_Name>;

SQL→ SELECT LENGTH (Ename) FROM EMP;

11) The ROUND () Function:

The ROUND () function is used to round a numeric field to the number of decimals specified.

Syntax:
SELECT ROUND (Column_Name, Decimals)
FROM EMP;

Parameter	Description
column_name	Required. The field to round.
decimals	Required. Specifies the number of decimals to be returned.

SQL→ SELECT ROUND (Comm, 2) FROM EMP;

Joins:

Orders Table

SR_NO	USER_ID	ITEM
1	2	Pizza
2	2	Burger
3	2	Milk Shake
4	3	French Fries
5	3	Burger
6	5	Diet Coke
7	5	Burger
8	99	Burger

Users Table

ID	NAME
1	Saif
2	Amik
3	Pankaj
4	Neeraj
5	Samadhan

```
CREATE TABLE Users
(
ID Number (5),
Name Varchar2 (15)
);

INSERT INTO Users (ID, Name) Values (1,'Saif');
INSERT INTO Users (ID, Name) Values (2,'Amik');
INSERT INTO Users (ID, Name) Values (3,'Pankaj');
INSERT INTO Users (ID, Name) Values (4,'Neeraj');
INSERT INTO Users (ID, Name) Values (5,'Samadhan');

CREATE TABLE Orders
(
Sr_No Number (5),
User_Id Number (5),
Item Varchar2 (15)
);

INSERT INTO Orders (Sr_No, User_Id, Item) Values (1,2,'Pizza');
INSERT INTO Orders (Sr_No, User_Id, Item) Values (2,2,'Burger');
INSERT INTO Orders (Sr_No, User_Id, Item) Values (3,2,'Milk Shake');
INSERT INTO Orders (Sr_No, User_Id, Item) Values (4,3,'French Fries');
INSERT INTO Orders (Sr_No, User_Id, Item) Values (5,3,'Burger');
INSERT INTO Orders (Sr_No, User_Id, Item) Values (6,5,'Diet Coke');
INSERT INTO Orders (Sr_No, User_Id, Item) Values (7,5,'Burger');
INSERT INTO Orders (Sr_No, User_Id, Item) Values (8,99,'Burger');
```

1) INNER JOIN (Default)

The ***inner join*** is the default used when you don't specify what type of Join. When you do an inner join of two tables it returns a new set of data with all of the instances of the join where the condition was met. If the condition was not met between the tables, the rows are ignored. This type of join will result in the smallest number of results.

SELECT * FROM Users
INNER JOIN Orders
ON Users.Id = Orders.User_Id;

The result is a combination of the two tables where the rows have been *joined* by their common *User_Id*. You can see that because Saif and Neeraj did not order anything, they did not show up in the results for this join. Additionally, user 99 who is in the order table, but not listed in the users table did not show up in the result either.

Id	Name	Sr_No	User_Id	Item
2	Amik	1	2	Pizza
2	Amik	2	2	Burger
2	Amik	3	2	Milk Shake
3	Pankaj	4	3	French Fries
3	Pankaj	5	3	Burger
5	Samadhan	6	5	Diet Coke
5	Samadhan	7	5	Burger

OUTER JOIN

Where an inner join ignores any rows that don't match the condition, an outer join can still show results if no condition was met. For example even though Saif and Neeraj did not order anything, they could still show up in the results of an Outer Join Query, depending on which table is favored. There are three options for favoring tables, and therefore there are three possible Outer Joins.

LEFT OUTER JOIN:

The left outer join means that the join will favor the left listed (first listed) table. Favoring the table means that all results from that table will be shown in the result, whether or not they match of the joined table on the condition. If they do not match any rows in the joined table, it will be attached to NULL Columns.

SELECT * FROM Users
LEFT OUTER JOIN Orders
ON Users.Id = Orders.User_Id
ORDER BY ID;

Id	Name	Sr_No	User_Id	Item
1	Saif	***NULL***	***NULL***	***NULL***
2	Amik	2	2	Burger
2	Amik	3	2	Milk Shake
2	Amik	1	2	Pizza
3	Pankaj	4	3	French Fries
3	Pankaj	5	3	Burger
4	Neeraj	***NULL***	***NULL***	***NULL***
5	Samadhan	7	5	Burger
5	Samadhan	6	5	Diet coke

Here we've listed the users table first, so it is the favored tables. As you can see Saif and Neeraj, who were not present in the results of the inner join as they did not order anything, are still listed in the results of this outer join. The columns that would normally be filled with values from the joined table are simply NULL.

RIGHT OUTER JOIN

The right outer join is the same as the left, except the favored table is the right listed (second listed) table.

```
SELECT * FROM Users
RIGHT OUTER JOIN Orders
ON Users.Id = Orders.User_Id
ORDER BY Sr_No;
```

Id	Name	Sr_No	User_Id	Item
2	Amik	1	2	Pizza
2	Amik	2	2	Burger
2	Amik	3	2	Milk Shake
3	Pankaj	4	3	French Fries
3	Pankaj	5	3	Burger
5	Samadhan	6	5	Diet Coke
5	Samadhan	7	5	Burger
NULL	***NULL***	8	99	Burger

You can see that by favoring the orders table we now see the unmatched User # 99 but not Saif and Neeraj.

FULL OUTER JOIN

A full outer join means that both tables are favored, so all rows from each table will be listed in the result regardless of whether they match any rows in the other table. In practice these joins are fairly rare. You can, however, replicate it easily with the UNION command by combining the results of the left outer and a right outer joins.

```
SELECT * FROM Users
FULL OUTER JOIN Orders
ON Users.Id = Orders.User_Id
ORDER BY Id;

SELECT * FROM Users LEFT OUTER JOIN Orders ON Users.Id = Orders.User_Id
UNION
SELECT * FROM Users RIGHT OUTER JOIN Orders ON Users.Id = Orders.User_Id;
```

Id	Name	Sr_No	User_Id	Item
1	Saif	***NULL***	***NULL***	***NULL***
2	Amik	1	2	Pizza
2	Amik	2	2	Burger
2	Amik	3	2	Milk Shake
3	Pankaj	4	3	French Fries
3	Pankaj	5	3	Burger
4	Neeraj	***NULL***	***NULL***	***NULL***
5	Samadhan	6	5	Diet Coke
5	Samadhan	7	5	Burger
NULL	***NULL***	8	99	Burger

As you can see, Saif and Neeraj are listed as well as the mysterious user # 99.

Rank: Let's assume we want to assign a sequential order, or rank, to people within a department based on salary, we might use the RANK function like.

SQL→SELECT EmpNo, Ename, Sal, DeptNo, RANK () OVER (ORDER BY Sal) AS Rank FROM EMP;

SQL→SELECT EmpNo, Ename, Sal, DeptNo, RANK () OVER (PARTITION BY DeptNo ORDER BY Sal) AS Rank FROM EMP;

Note: What we see here is where two people have the same salary they are assigned the same rank. When multiple rows share the same rank the next rank in the sequence is not consecutive.

DENSE_RANK: The DENSE_RANK function acts like the RANK function except that it assigns consecutive ranks

SQL→SELECT EmpNo, Ename, Sal, DeptNo, DENSE_RANK () OVER (ORDER BY Sal) AS Rank FROM EMP;

SQL→ SELECT EmpNo, Ename, Sal, DeptNo, DENSE_RANK () OVER (PARTITION BY DeptNo ORDER BY Sal) AS Rank FROM EMP;

Sequence:

Use the CREATE SEQUENCE statement to create a sequence, which is a database object from which multiple users may generate unique integers. You can use sequences to automatically generate primary key values.

When a sequence number is generated, the sequence is incremented, independent of the transaction committing or rolling back.

Once a sequence is created, you can access its values in SQL statements with the CURRVAL Pseudo Column, which returns the current value of the sequence, or the NEXTVAL Pseudo Column, which increments the sequence and returns the new value.

Prerequisites:

To create a sequence in your own schema, you must have the CREATE SEQUENCE system privilege.
To create a sequence in another user's schema, you must have the CREATE ANY SEQUENCE system privilege.

Syntax:

CREATE SEQUENCE <Sequence_Name>
Start with Value
Increment by Value
MINVALUE Value
MAXVALUE Value
CYCLE/NOCYCLE
CACHE Value;

Let's cover each of their parameters:

1) Start With – Defines the first number of the sequence. Default is one.

2) Increment By – Defines how many to add to get the next value. Default is one.

3) Minvalue – Defines the lowest value of the Sequence when the Sequence is created to count down (Increment by a minus number)

4) Maxvalue – Defines the largest value of the Sequence. Default is 10E23.

5) Cycle/Nocycle – Tells the Sequence to start over once it reaches the Max Value or Min Value. The default is Nocycle.

6) Cache/NoCache – Tells the database how many numbers to Cache in memory. The default is 20. So, if the Sequence is starting at one, the database will cache 1…20 and set the sequence at 21. The database will answer the NEXTVAL request from the numbers in memory. Once those numbers are used, the database will load 20 more numbers in memory. If the database is shutdown, all sequence numbers in memory will be lost. When the database restarts, it will look at the number for the sequence and load the next 20 into memory,

setting the sequence forward by 20. This improves response but will lose some numbers. Setting the NOCACHE will cause the database not to cache any numbers in memory.

CREATE SEQUENCE Saif
Start With 1
Increment By 1
Cache 20;

To check Next Value from the Sequence:
Syntax:
SELECT Saif.NextVal FROM EMP;
SELECT Saif.NextVal FROM Dual;

To check Current Value from the Sequence:
Syntax:
SELECT Saif.CurrVal FROM EMP;
SELECT Saif.CurrVal FROM Dual;

CREATE SEQUENCE My_Seq
Start With 1
Increment By 1
MaxValue 14
Cycle
Cache 10;

SELECT My_Seq.Nextval AS Saif, EMP.* FROM EMP;

To check Next Value from the Sequence:
Syntax:
SELECT My_Seq.NextVal FROM EMP;
SELECT My_Seq.NextVal FROM Dual;

To check Current Value from the Sequence:
Syntax:
SELECT My_Seq.Currval FROM EMP;
SELECT My_Seq.CurrVal FROM Dual;

Drop Sequence:

Syntax:
DROP Sequence <Sequence_Name>

DROP Sequence Saif;
DROP Sequence My_Seq;

To Find Sequences created by User:
SQL→SELECT * FROM USER_SEQUENCES;

Database Queries:

1) How to find all details about Constraints?
SQL → SELECT * From User_Constraints;
SQL → SELECT * FROM User_Cons_Columns;

2) How to find Constraint Name?
SQL→SELECT Table_Name, Constraint_Name FROM User_Constraints;

3) How to find Constraint Name with Column_Name?
SQL→SELECT Column_Name, Table_Name, Constraint_Name FROM User_Cons_Columns;

5) How to find Selected Tables from a User?
SQL→SELECT Table_Name FROM User_Tables WHERE Table_Name LIKE 'STU%';

6) How to find Selected Tables which have Constraint?
SQL→SELECT Table_Name FROM User_Cons_Columns WHERE Table_Name LIKE 'STU%';

7) How to find Constraint_Name, Constraint_Type, Table_Name?
SQL →SELECT Table_Name, Constraint_Type, Constraint_Name FROM User_Constraints;
SQL →SELECT Table_Name, Constraint_Type, Constraint_Name, Generated FROM User_Constraints;

8) How to Select Users from Database?
SQL→SELECT Username FROM All_Users ORDER BY Username;

9) How to check Sequences?
SQL→SELECT * FROM USER_SEQUENCES;

10) How to check Procedures?
SQL→ SELECT * FROM User_Source
WHERE Type='PROCEDURE'
AND NAME IN ('SP_CONNECTED_AGG','SP_UNCONNECTED_AGG');

11) How to find DB Name?
SQL→ SELECT Ora_Database_Name FROM DUAL;
SELECT * FROM GLOBAL_NAME;
SELECT Name from V$DATABASE;