Module: SQL

Module Overview

SQL is the query language—it’s used for accessing, cleaning, and analyzing data that are stored in databases. It’s very easy to learn, yet it’s employed by the world’s largest companies to solve incredibly challenging problems.
This module will aim at making you understand and implement the SQL queries.

Module Objective

At the end of the module, you will be able,

- To understand core database concepts
- To understand how data is stored in tables
- To understand DDL, DCL, DML commands
- To know the sorting of data using SQL
- To know the functions in SQL
- To understand the essential topics such as Sub-queries, Joins, views, unions, indexes, etc.
- To understand triggers and cursors

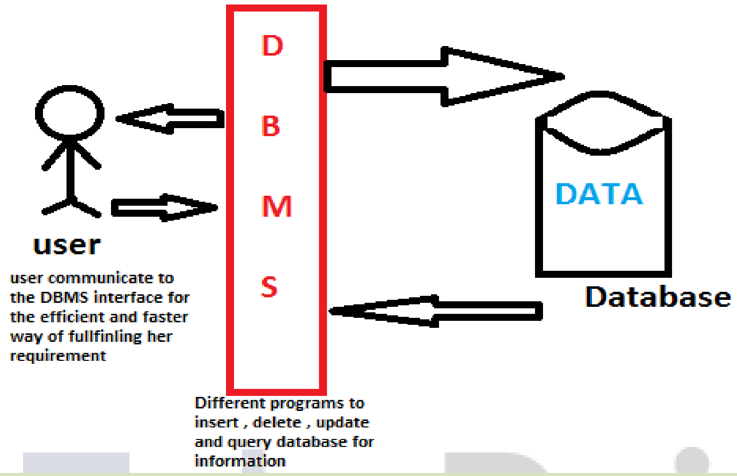
Database fundamentals and SQL Basics

What is a Database?

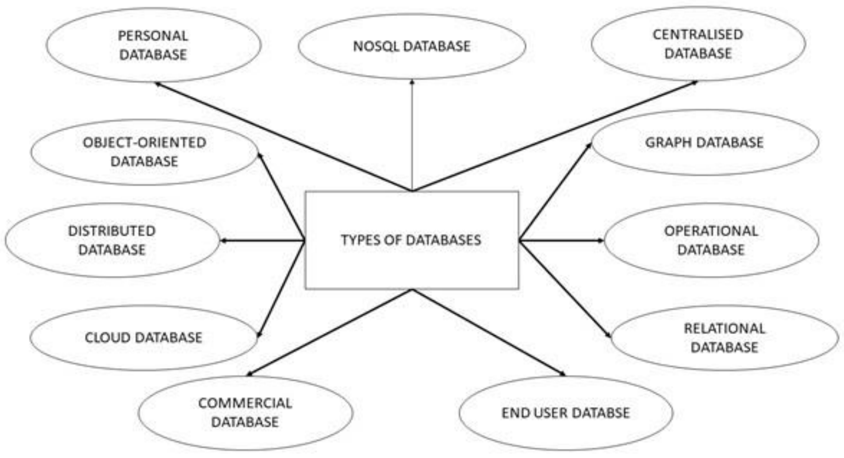
- ➢ Database is a systematic collection of data. Databases support storage and manipulation of data. Databases make data management easy.
- ➢ For example, Let's consider the facebook. It needs to store, manipulate and present data related to members, their friends, member activities, messages, advertisements and lot more. Hence, it uses database for the same.

What is a Database Management System (DBMS)?

- ➢ Database Management System (DBMS) is a collection of programs which enables its users to access database, manipulate data, reporting / representation of data .


D
B
M
S
DATA
user
Database
user communicate to the DBMS interface for the efficient and faster way of fullfinling her requirement
Different programs to insert , delete , update and query database for information

Types of DBMS

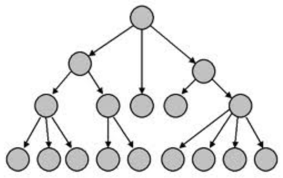


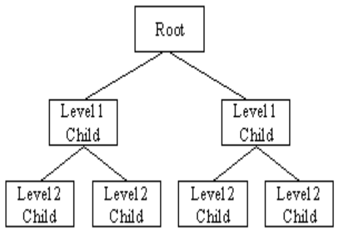
Depending upon the usage requirements, the figure depicts various types of databases available in the market. Few of the databases are explained below

➢ **Hierarchical** –

- This type of DBMS employs the "parent-child" relationship of storing data. This type of DBMS is rarely used nowadays. Its structure is like a tree with nodes representing records and branches representing fields.In the Hierarchical Database Model we have to learn about the databases.
- It is very fast and simple.
- The structure implies that a record can have also a repeating information.
- In this structure Data follows a series of records, It is a set of field values attached to it. It collects all records together as a record type.

- These record types are the equivalent of tables in the relational model, and with the individual records being the equivalent of rows. To create links between these record types, the hierarchical model uses

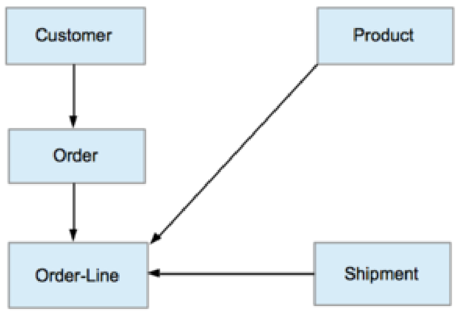




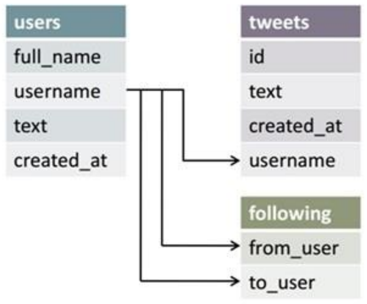
  these type Relationships.
- **Advantage**
  Hierarchical database can be accessed and updated rapidly because in this model structure is like as a tree and the relationships between records are defined in advance. This feature is a two-edged.
- **Disadvantage**
  This type of database structure is that each child in the tree may have only one parent, and relationships or linkages between children are not permitted, even if they make sense from a logical standpoint. Hierarchical databases are so in their design. it can adding a new field or record requires that the entire database be redefined.

➢ **Network DBMS** –

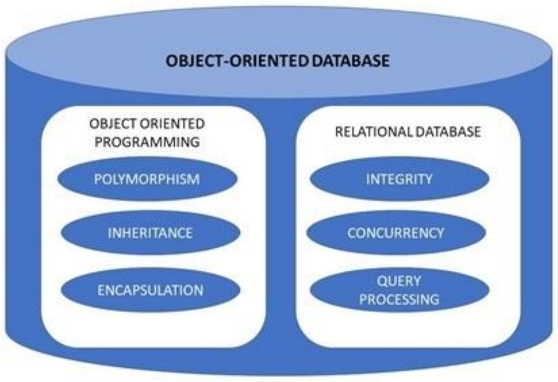
- This type of DBMS supports many-to many relations. This usually results in complex database structures.
- A network databases are mainly used on a large digital computers.
- It more connections can be made between different types of data, network databases are considered more efficiency It contains limitations must be considered when we have to use this kind of database.
- It is Similar to the hierarchical databases, network databases.
- Network databases are similar to hierarchical databases by also having a hierarchical structure.
- A network database looks more like a cobweb or interconnected network of records.
- In network databases, children are called **members** and parents are called **occupier**.
- The difference between each child or member can have more than one parent.



- **Relational DBMS –**
  - This type of DBMS defines database relationships in form of tables, also known as **relations**. This is the most popular DBMS type in the market. Examples of relational database management systems include MySQL, Oracle, and Microsoft SQL Server database.
  - These databases are categorized by a set of tables where data gets fit into a pre-defined category.
  - The table consists of rows and columns where the column has an entry for data for a specific category and rows contains instance for that data defined according to the category.
  - The Structured Query Language (SQL) is the standard user and application program interface for a relational database.
  - There are various simple operations that can be applied over the table which makes these databases easier to extend, join two databases with a common relation and modify all existing applications.



- **Object Oriented Relation DBMS** –
  - This type supports storage of new data types.
  - The data to be stored is in form of **objects**.
  - The objects to be stored in the database have attributes (i.e. gender, ager) and methods that define what to do with the data.
  - **PostgreSQL** is an example of an object oriented relational DBMS.
  - An object-oriented database is a collection of object-oriented programming and relational database. There are various items which are created using object-oriented programming languages like C++, Java which can be stored in relational databases, but object-oriented databases are well-suited for those items.
  - An object-oriented database is organized around objects rather than actions, and data rather than logic. For example, a multimedia record in a relational database can be a definable data object, as opposed to an alphanumeric value.



What is SQL?

- Structured Query language (SQL) **pronounced as "S-Q-L" or sometimes as "See-Quel"**is actually the standard language for dealing with Relational Databases.
- SQL programming can be effectively used to insert, search, update, delete database records.Relational databases like MySQL Database, Oracle, Ms SQL server, Sybase, etc uses SQL.
- SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987.

What is NoSQL ?

- The main characteristic of NoSQL is its non-adherence to Relational Database Concepts. NOSQL means "Not only SQL".
- Concept of NoSQL databases grew with internet giants such as Google, Facebook, Amazon etc who deal with gigantic volumes of data.
- When you use relational database for massive volumes of data , the system starts getting slow in terms of response time. To overcome this , we could of course "scale up" our systems by upgrading our existing hardware.
- The alternative to the above problem would be to distribute our database load on multiple hosts as the load increases.This is known as "scaling out".
- NOSQL database are non-relational databases that scale out better than relational databases and are designed with web applications in mind.
- NOSQL database do not use SQL to query the data and do not follow strict schemas like relational models.With NoSQL, ACID (Atomicity, Consistency, Isolation, Durability) features are not guaranteed always.
- NoSQL databases are especially useful for working with large sets of distributed data.
- NoSQL database examples include MongoDB, BigTable, Redis, RavenDB Cassandra, HBase, Neo4j and CouchDB.

What is MySQL?

- MySQL is a freely available open source Relational Database Management System (RDBMS) that uses Structured Query Language (SQL).

- Its name is a combination of "My", the name of co-founder Michael Widenius's daughter, and "SQL", the abbreviation for Structured Query Language.
- MySQL is a central component of the **LAMP** open-source web application software stack (and other "AMP" stacks).
- LAMP is an acronym for "Linux, Apache, MySQL, Perl/PHP/Python".
- Applications that use the MySQL database include: **TYPO3**, **MODx, Joomla, WordPress**, **Simple Machines Forum, phpBB, MyBB,** and**Drupal**.
- MySQL is also used in many high-profile, large-scale websites, including **Google** (though not for searches), **Facebook**, **Twitter**,**Flickr** and **YouTube**.
- **MySQL** is a freely available open source Relational Database Management System (RDBMS) that uses Structured Query Language (**SQL**).
- MySQL runs on virtually all platforms, including Linux, UNIX and Windows.

Why we Use MySQL?

- There are a number of relational database management systems on the market.Examples of relational databases include Microsoft SQL Server, Microsoft Access, Oracle, DB2 etc.

But MySQL is widely used due to following reasons:

- ✓ high performance
- ✓ Cost effective
- ✓ Cross platform

RDBMS Concepts

- RDBMS stands for Relational Database Management System.
- RDBMS is the basis forSQL, 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 therelational model as introduced by **E. F. Codd**.

What is a table?

- The data in an RDBMS is stored in database objects which are called as tables.
- This table is basically a collection of related data entries and it consists of numerous columns and rows.
- "**Table**" is another term for "**relation**"

The following program is an example of a CUSTOMERS table –

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
```

What is a Record or a Row?

- A record is also called as 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 | Ramesh   |  32 | Ahmedabad |  2000.00 |
+----+----------+-----+-----------+----------+
```

A record is a horizontal entity in a table.

What is a 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 be as shown below −

```
+-----------+
| ADDRESS   |
+-----------+
| Ahmedabad |
| Delhi     |
| Kota      |
| Mumbai    |
| Bhopal    |
| MP        |
| Indore    |
+----+------+
```

What is a 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 withno 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 the one that has been left blank during a record creation.

What is Normalization?

- Database normalization is the process of efficiently organizing data in a database. There are two reasons of
  this normalization process –
- Eliminating redundant data, for example, storing the same data in more than one table.
- Ensuring data dependencies make sense.
- It divides larger tables to smaller tables and links them using relationships.
- Normalization rules are divided into the following normal forms:
  - First Normal Form
  - Second Normal Form
  - Third Normal Form
  - BCNF

1. **First Normal Form**
   - Eliminate repeating groups in individual tables.
   - Create a separate table for each set of related data.

- Identify each set of related data with a primary key.

Do not use multiple fields in a single table to store similar data.

2. **Second Normal Form**
   - Create separate tables for sets of values that apply to multiple records.
   - Relate these tables with a foreign key.

Records should not depend on anything other than a table's primary key (a compound key, if necessary).

3. **Third Normal Form**
   - Eliminate fields that do not depend on the key.

Values in a record that are not part of that record's key do not belong in the table. In general, any time the contents of a group of fields may apply to more than a single record in the table, consider placing those fields in a separate table.

4. **Other Normalization Forms**
   - Fourth normal form, also called Boyce Codd Normal Form (BCNF), and fifth normal form do exist, but are rarely considered in practical design.
   - Disregarding these rules may result in less than perfect database design, but should not affect functionality.

Normalizing an Example Table

These steps demonstrate the process of normalizing a fictitious student table.

1. **Unnormalized table:**

Student#	Advisor	Adv-Room	Class1	Class2	Class3
1022	Jones	412	101-07	143-01	159-02
4123	Smith	216	201-01	211-02	214-01

2. **First Normal Form: No Repeating Groups**

   Tables should have only two dimensions. Since one student has several classes, these classes should be listed in a separate table. Fields Class1, Class2, and Class3 in the above records are indications of
   design trouble.

Spreadsheets often use the third dimension, but tables should not. Another way to look at this problem is with a one-to-many relationship, do not put the one side and the many side in the same table. Instead, create another table in first normal form by eliminating the repeating group (Class#),
as shown below:

Student#	Advisor	Adv-Room	Class#
1022	Jones	412	101-07
1022	Jones	412	143-01
1022	Jones	412	159-02
4123	Smith	216	201-01
4123	Smith	216	211-02
4123	Smith	216	214-01

3. **Second Normal Form: Eliminate Redundant Data**

   Note the multiple Class# values for each Student# value in the above table. Class# is not functionally dependent on Student# (primary key), so this relationship is not in second normal form.

   The following two tables demonstrate second normal form:

   Students:

Student#	Advisor	Adv-Room
1022	Jones	412
4123	Smith	216

Registration:

Student#	Class#
1022	101-07
1022	143-01
1022	159-02
4123	201-01
4123	211-02
4123	214-01

4. **Third Normal Form: Eliminate Data Not Dependent On Key**

   In the last example, Adv-Room (the advisor's office number) is functionally dependent on the Advisor attribute. The solution is to move that attribute from the Students table to the Faculty table,
   as shown below:

   Students:

Student#	Advisor
1022	Jones
4123	Smith

Faculty:

Name	Room	Dept
Jones	412	42
Smith	216	42

SQL: Datatypes

Database Tables

A database most often contains one or more tables. Each table is identified by a name (e.g. "Customers" or "Orders"). Tables contain records (rows) with data.

Tips : SQL keywords **are NOT case sensitive**: select is the same as SELECT

Semicolon after SQL Statements?

Some database systems require a semicolon at the end of each SQL statement.

Semicolon is the standard way to separate each SQL statement in database systems that allow more than one SQL statement to be executed in the same call to the server.

DATA TYPES

Data types define the nature of the data that can be stored in a particular column of a table

MySQL has **3** main categories of data types namely

1. Numeric,
2. Text
3. Date/time.

1.Numeric Data types

Numeric data types are used to store numeric values. It is very important to make sure range of your data is between lower and upper boundaries of numeric data types.

TINYINT( )	-128 to 127 normal 0 to 255 UNSIGNED.

SMALLINT( )	-32768 to 32767 normal 0 to 65535 UNSIGNED.
MEDIUMINT( )	-8388608 to 8388607 normal 0 to 16777215 UNSIGNED.
INT( )	-2147483648 to 2147483647 normal 0 to 4294967295 UNSIGNED.
BIGINT( )	-9223372036854775808 to 9223372036854775807 normal 0 to 18446744073709551615 UNSIGNED.
FLOAT	A small approximate number with a floating decimal point.
DOUBLE(,)	A large number with a floating decimal point.
DECIMAL(,)	A DOUBLE stored as a string , allowing for a fixed decimal point. Choice for storing currency values.

2.Text Data Types

As data type category name implies these are used to store text values. Always make sure you length of your textualdata do not exceed maximum lengths.

CHAR( )	A fixed section from 0 to 255 characters long.
VARCHAR( )	A variable section from 0 to 255 characters long.

TINYTEXT	A string with a maximum length of 255 characters.
TEXT	A string with a maximum length of 65535 characters.
BLOB	A string with a maximum length of 65535 characters.
MEDIUMTEXT	A string with a maximum length of 16777215 characters.
MEDIUMBLOB	A string with a maximum length of 16777215 characters.
LONGTEXT	A string with a maximum length of 4294967295 characters.
LONGBLOB	A string with a maximum length of 4294967295 characters.

3.Date / Time

DATE	YYYY-MM-DD
DATETIME	YYYY-MM-DD HH:MM:SS
TIMESTAMP	YYYYMMDDHHMMSS
TIME	HH:MM:SS

Apart from above there are some other data types in MySQL.

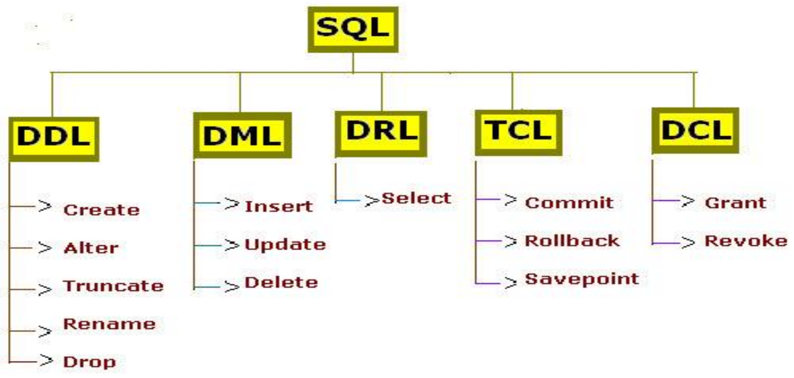
ENUM	To store text value chosen from a list of predefined text values
SET	This is also used for storing text values chosen from a list of predefined text values. It can have multiple values.
BOOL	Synonym for TINYINT(1), used to store Boolean values
BINARY	Similar to CHAR, difference is texts are stored in binary format.
VARBINARY	Similar to VARCHAR, difference is texts are stored in binary format.

SQL: Commands

SQL Commands

SQL commands are mainly categorized into five categories as discussed below:

- DDL
- DML
- DRL
- DCL
- TCL



DDL(Data Definition Language) :

DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in database.

Examples of DDL commands:

1)CREATE – is used to create the database or its objects (like table, index, function, views, store procedure and triggers). There are two CREATE statements available in SQL:

1. CREATE DATABASE
2. CREATE TABLE

i.CREATE DATABASE

A Database is defined as a structured set of data. So, in SQL the very first step to store the data in a well structured manner is to create a database. The CREATE DATABASEstatement is used to create a new database in SQL.

Syntax:

```
CREATE DATABASE database_name;


database_name: name of the database.
```

Example:

This query will create a new database in SQL and name the database as my_database.

```
CREATE DATABASE my_database;
```

ii.CREATE TABLE

The CREATE TABLE statement is used to create a table in SQL. We know that a table comprises of rows and columns. So while creating tables we have to provide all the information to SQL about the names of the columns, type of data to be stored in columns, size of the data etc. Let us now dive into details on how to use CREATE TABLE statement to create tables in SQL.

Syntax:

```
CREATE TABLE table_name

(

column1 data_type(size),

column2 data_type(size),

column3 data_type(size),

....
);


table_name:  name of the table.
column1 name of the first column.
data_type: Type of data we want to store in the particular column.
For example,int for integer data.
size: Size of the data we can store in a particular column.
For example if for a column we specify the data_type as int and size as 10 then this column can store an integer number of maximum 10 digits.
```

ExampleQuery:
This query will create a table named Students with three columns, ROLL_NO, NAME and SUBJECT.

```
CREATE TABLE Students

(

ROLL_NO int(3),

NAME varchar(20),

SUBJECT varchar(20),

);
```

2)DROP – is used to delete objects from the database.
Examples:

```
DROP TABLE table_name;
table_name: Name of the table to be deleted.

DROP DATABASE database_name;
database_name: Name of the database to be deleted.
```

If you want to delete an existing database <test>, then the DROP DATABASE statement would be as shown below −

```
SQL> DROP DATABASE testDB;
```

NOTE − Be careful before using this operation because by deleting an existing database would result in loss of complete information stored in the database.

Similarly, use delete a table in database using DROP.

3)ALTER-is used to alter the structure of the database.

ALTER TABLE is used to add, delete/drop or modify columns in the existing table. It is also used to add and drop various constraints on the existing table.

i.ALTER TABLE – ADD

ADD is used to add columns into the existing table.

Syntax:

```
ALTER TABLE table_name
        ADD (Columnname_1  datatype,
        Columnname_2  datatype,
        …
        Columnname_n  datatype);
```

ii.ALTER TABLE – DROP

DROP COLUMN is used to drop column in a table. Deleting the unwanted columns from the table.

Syntax:

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

iii.ALTER TABLE-MODIFY

It is used to modify the existing columns in a table. Multiple columns can also be modified at once.

Syntax:

```
ALTER TABLE table_name
MODIFY column_name column_type;
```

Example

Consider the CUSTOMERS table having the following records –

```
+----+----------+-----+-----------+----------+

| ID | NAME     | AGE | ADDRESS   | SALARY   |

+----+----------+-----+-----------+----------+

|1|Ramesh|32|Ahmedabad|2000.00|

|2|Khilan|25|Delhi|1500.00|

|3| kaushik  |23|Kota|2000.00|
|4|Chaitali|25|Mumbai|6500.00|

|5|Hardik|27|Bhopal|8500.00|

|6|Komal|22| MP       |4500.00|

|7|Muffy|24|Indore|10000.00|

+----+----------+-----+-----------+----------+
```

Following is the example to ADD a New Column to an existing table −

```
ALTER TABLE CUSTOMERS ADD SEX char(1);
```

Now, the CUSTOMERS table is changed and following would be output from the SELECT statement.

```
+----+---------+-----+-----------+----------+------+
| ID | NAME    | AGE | ADDRESS   | SALARY   | SEX  |
+----+---------+-----+-----------+----------+------+
|  1 | Ramesh  |  32 | Ahmedabad |  2000.00 | NULL |
|  2 | Ramesh  |  25 | Delhi     |  1500.00 | NULL |
|  3 | kaushik |  23 | Kota      |  2000.00 | NULL |
|  4 | kaushik |  25 | Mumbai    |  6500.00 | NULL |
|  5 | Hardik  |  27 | Bhopal    |  8500.00 | NULL |
|  6 | Komal   |  22 | MP        |  4500.00 | NULL |
|  7 | Muffy   |  24 | Indore    | 10000.00 | NULL |
+----+---------+-----+-----------+----------+------+
```

Following is the example to DROP sex column from the existing table.

```
ALTER TABLE CUSTOMERS DROP SEX;
```

Now, the CUSTOMERS table is changed and following would be the output from the SELECT statement.

```
+----+---------+-----+-----------+----------+
| ID | NAME    | AGE | ADDRESS   | SALARY   |
+----+---------+-----+-----------+----------+
|  1 | Ramesh  |  32 | Ahmedabad |  2000.00 |
|  2 | Ramesh  |  25 | Delhi     |  1500.00 |
|  3 | kaushik |  23 | Kota      |  2000.00 |
|  4 | kaushik |  25 | Mumbai    |  6500.00 |
|  5 | Hardik  |  27 | Bhopal    |  8500.00 |
|  6 | Komal   |  22 | MP        |  4500.00 |
|  7 | Muffy   |  24 | Indore    | 10000.00 |
+----+---------+-----+-----------+----------+
```

4)TRUNCATE–is used to remove all records from a table, including all spaces allocated for the records are removed.

Syntax:

```
TRUNCATE TABLE  table_name;

table_name: Name of the table to be truncated.
```

RENAME Command –It is used to rename an object existing in the database.
Syntax:

```
ALTER TABLE table_name
RENAME TO new_table_name;
```

DML(Data Manipulation Language) :

The SQL commands that deals with the manipulation of data present in database belong to DML or Data Manipulation Language and this includes most of the SQL statements.
DML changes data in an object. If you insert a row into a table, that is DML.
All DML statements change data, and must be committed before the change becomes permanent.

Examples of DML:
1)SELECT – is used to retrieve data from the a database.
Basic Syntax:

```
SELECT column1,column2 FROM table_name
column1 , column2: names of the fields of the table

table_name: from where we want to fetch
```

This query will return all the rows in the table with fields column1 , column2.

- To fetch the entire table or all the fields in the table:

```
SELECT * FROM table_name;
```

- Query to fetch the fields ROLL_NO, NAME, AGE from the table Student:

```
SELECT ROLL_NO, NAME, AGE FROM Student;
```

2)INSERT – is used to insert data into a table.
There are two ways of using INSERT INTO statement for inserting rows:

Only values: First method is to specify only the value of data to be inserted without the column names.
Syntax:

```
INSERT INTO table_name VALUES (value1, value2, value3,...);

table_name: name of the table.
value1, value2,.. : value of first column, second column,... for the new record
```

Column names and values both: In the second method we will specify both the columns which we want to fill and their corresponding values as shown below:
Syntax:

```
INSERT INTO table_name (column1, column2, column3,..) VALUES ( value1, value2, value3,..);

table_name: name of the table.
column1: name of first column, second column ...
```

value1, value2, value3 : value of first column, second column,... for the new record

3)UPDATE – is used to update existing data within a table.
Basic Syntax

```
UPDATE table_name SET column1 = value1, column2 = value2,...
WHERE condition;

table_name: name of the table
column1: name of first , second, third column....
value1: new value for first, second, third column....
condition: condition to select the rows for which the
values of columns needs to be updated.
```

4)DELETE – is used to delete records from a database table.
Basic Syntax:

```
DELETE FROM table_name WHERE some_condition;

table_name: name of the table

some_condition: condition to choose particular record.
```

Difference between Delete,Drop and Truncate:

DELETE Statement:
The DELETE command is used to remove rows from a table. A WHERE clause can be used to only remove some rows. If no WHERE condition is specified, all rows will be removed. After performing a DELETE operation you need to COMMIT or ROLLBACK the transaction to make the change permanent or to undo it. Note that this operation will cause all DELETE triggers on the table to fire.
TRUNCATE statement:
TRUNCATE removes **all rows** from a table. The operation cannot be rolled back and no triggers will be fired.

DROPstatement:
The DROP command removes a table from the database. All the tables' rows, indexes and privileges will also be removed. No DML triggers will be fired. The operation cannot be rolled back.

DRL/DSL(Data Retrieval Language/ Data Selection Language) :

- DRL/DSL stands for Data Retrieval Language/Data Selection Language.
- It is a set commands which are used to retrieve data from database server.
- It manipulates the data in database for display purpose like aggregate function.
- In DRL/DSL, for accessing the data it uses the DML command that is SELECT.
- The SELECT command allows database users to retrieve the specific information they desire from an operational database.

SELECT clause has many optional clauses are as follow;

Clause	**Description**
FROM	It is used for selecting a table name in a database.
WHERE	It specifies which rows to retrieve.
GROUP BY	It is used to arrange the data into groups.
HAVING	It selects among the groups defined by the GROUP BY clause.
ORDER BY	It specifies an order in which to return the rows.
AS	It provides an alias which can be used to temporarily rename tables or columns.

Syntax:
SELECT {* | column_name1, column_name2, . . . , column_name_n}
FROM <list_of_tablename>
WHERE <condition>;

Example

```
SELECT * FROM employee
WHERE salary >=10000;


OR

SELECT eid, ename, age, salary
WHERE salary >=10000;
```

DCL(Data Control Language) :

DCL includes commands such as GRANT and REVOKE which mainly deals with the rights, permissions and other controls of the database system.

Examples of DCL commands:
1)GRANT-gives user's access privileges to database.
2)REVOKE-withdraw user's access privileges given by using the GRANT command.

- **Allow a User to create session**

When we create a user in SQL, it is not even allowed to login and create a session until and unless proper permissions/priviliges are granted to the user.

Following command can be used to grant the session creating priviliges.

```
GRANT CREATE SESSION TO username;
```

- **Allow a User to create table**

To allow a user to create tables in the database, we can use the below command,

```
GRANT CREATE TABLE TO username;
```

- **Provide user with space on tablespace to store table**

Allowing a user to create table is not enough to start storing data in that table. We also must provide the user with priviliges to use the available tablespace for their table and data.

```
ALTER USER username QUOTA UNLIMITED ON SYSTEM;
```

The above command will alter the user details and will provide it access to unlimited tablespace on system.

NOTE: Generally unlimited quota is provided to Admin users.

- Grant all privilege to a User

sysdba is a set of priviliges which has all the permissions in it. So if we want to provide all the privileges to any user, we can simply grant them the sysdba permission.

```
GRANT sysdba TO username
```

- **Grant permission to create any table**

Sometimes user is restricted from creating come tables with names which are reserved for system tables. But we can grant privileges to a user to create any table using the below command,

```
GRANT CREATE ANY TABLE TO username
```

- **Grant permission to drop any table**

As the title suggests, if you want to allow user to drop any table from the database, then grant this privilege to the user,

```
GRANT DROP ANY TABLE TO username
```

- **To take back Permissions**

And, if you want to take back the privileges from any user, use the REVOKE command.

```
REVOKE CREATE TABLE FROM username
```

TCL(transaction Control Language) :

TCL commands deals with the transaction within the database.

Examples of TCL commands:
1) COMMIT– commits a Transaction.
When we use any DML command like INSERT, UPDATE or DELETE, the changes made by these commands are not permanent, until the current session is closed, the changes made by these commands can be rolled back.

To avoid that, we use the COMMIT command to mark the changes as permanent.

Following is commit command's syntax,

```
COMMIT;
```

2)ROLLBACK– rollbacks a transaction in case of any error occurs.
This command restores the database to last commited state. It is also used with SAVEPOINT command to jump to a savepoint in an ongoing transaction.

If we have used the UPDATE command to make some changes into the database, and realise that those changes were not required, then we can use the ROLLBACK command to rollback those changes, if they were not commited using the COMMIT command.

Following is rollback command's syntax,

```
ROLLBACK TO savepoint_name;
```

3)SAVEPOINT–sets a savepoint within a transaction.
SAVEPOINT command is used to temporarily save a transaction so that you can rollback to that point whenever required.

Following is savepoint command's syntax,

```
SAVEPOINT savepoint_name;
```

In short, using this command we can name the different states of our data in any table and then rollback to that state using the ROLLBACK command whenever required.

Using Savepoint and Rollback

Following is the table class,

id	name
1	Abhi
2	Adam
4	Alex

Lets use some SQL queries on the above table and see the results.

```
INSERT INTO class VALUES(5, 'Rahul');

COMMIT;

UPDATE class SET name = 'Abhijit' WHERE id = '5';

SAVEPOINT A;

INSERT INTO class VALUES(6, 'Chris');

SAVEPOINT B;

INSERT INTO class VALUES(7, 'Bravo');

SAVEPOINT C;

SELECT * FROM class;
```

NOTE: SELECT statement is used to show the data stored in the table.

The resultant table will look like,

id	name
1	Abhi

2	Adam
4	Alex
5	Abhijit
6	Chris
7	Bravo

Now let's use the ROLLBACK command to roll back the state of data to the savepoint B.

```
ROLLBACK TO B;
SELECT * FROM class;
```

Now our class table will look like,

id	name
1	Abhi
2	Adam
4	Alex
5	Abhijit

6	Chris

Now let's again use the ROLLBACK command to roll back the state of data to the savepoint A

```
ROLLBACK TO A;

SELECT * FROM class;
```

Now the table will look like,

id	name
1	Abhi
2	Adam
4	Alex
5	Abhijit

So now you know how the commands COMMIT, ROLLBACK and SAVEPOINT works.

Lab Activity:
Students kindly perform the above discussed topics in your lab sessions.

Operators

SQL : OPERATORS

- An operator is a reserved word or a character used primarily in an SQL statement's WHERE clause to perform operation(s), such as comparisons, logical and arithmetic operations.
- These Operators are used to specify conditions in an SQL statement and to serve as conjunctions for multiple conditions in a statement.
- Below mentioned are the various operators used -
  - ❖ Arithmetic operators
  - ❖ Comparison operators
  - ❖ Logical operators

1. SQL Arithmetic Operators:

Assume 'variable a' holds 10 and 'variable b' holds 20, then –

Show Examples

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator.	a + b will give 30
- (Subtraction)	Subtracts right hand operand from left hand operand.	a - b will give -10
* (Multiplication)	Multiplies values on either side of the operator.	a * b will give 200
/ (Division)	Divides left hand operand by right hand operand.	b / a will give 2

% (Modulus)	Divides left hand operand by right hand operand and returns remainder.	b % a will give 0

2. SQL Comparison Operators

Assume 'variable a' holds 10 and 'variable b' holds 20, then −

Show Examples

Operator	Description	Example
=	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(a = b) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a != b) is true.
<>	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a <> b) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(a > b) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(a < b) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(a >= b) is not true.

<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(a <= b) is true.
!<	Checks if the value of left operand is not less than the value of right operand, if yes then condition becomes true.	(a !< b) is false.
!>	Checks if the value of left operand is not greater than the value of right operand, if yes then condition becomes true.	(a !> b) is true.

3. SQL Logical Operators

Here is a list of all the logical operators available in SQL.

Show Examples

Sr.No.	Operator & Description
1	ALL The ALL operator is used to compare a value to all values in another value set.
2	AND The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
3	ANY The ANY operator is used to compare a value to any applicable value in the list as per the condition.
4	BETWEEN

	The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.
5	EXISTS The EXISTS operator is used to search for the presence of a row in a specified table that meets a certain criterion.
6	IN The IN operator is used to compare a value to a list of literal values that have been specified.
7	LIKE The LIKE operator is used to compare a value to similar values using wildcard operators.
8	NOT The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. This is a negate operator.
9	OR The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.
10	IS NULL The NULL operator is used to compare a value with a NULL value.
11	UNIQUE

	The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates).

Examples:

AND Example : The following SQL statement selects all fields from "Customers" where country is "Germany" AND city is “Berlin”.

```
SELECT * FROM Customers
WHERE Country='Germany' AND City='Berlin';
```

OR Example: The following SQL statement selects all fields from "Customers" where city is "Berlin" OR "München":

```
SELECT * FROM Customers
WHERE City='Berlin' OR City='München';
```

NOT Example: The following SQL statement selects all fields from "Customers" where country is NOT "Germany":

```
SELECT * FROM Customers
WHERE NOT Country='   Germany';
```

Similarly, students work with the other operators in your lab sessions.

Lab Activity:
Students kindly perform the above discussed topics in your lab sessions.

Expressions

An expression is a combination of one or more values, operators and SQL functions that evaluate to a value. These SQL EXPRESSIONs are like formulae and they are written in query language. You can also use them to query the database for a specific set of data.

Consider the basic syntax of the SELECT statement as follows −

```
SELECT column1, column2, columnN
FROM table_name
WHERE [CONDITION|EXPRESSION];
```

There are different types of SQL expressions, which are mentioned below −

- Boolean
- Numeric
- Date

1. Boolean Expressions

SQL Boolean Expressions fetch the data based on matching a single value. Following is the syntax −

```
SELECT column1, column2, columnN
FROM table_name
WHERE SINGLE VALUE MATCHING EXPRESSION;
```

Consider the CUSTOMERS table having the following records −

```
SQL> SELECT * FROM CUSTOMERS;

+----+----------+-----+-----------+----------+

| ID | NAME     | AGE | ADDRESS   | SALARY   |

+----+----------+-----+-----------+----------+

|1|Ramesh|32|Ahmedabad|2000.00|

|2|Khilan|25|Delhi|1500.00|

|3| kaushik  |23|Kota|2000.00|

|4|Chaitali|25|Mumbai|6500.00|

|5|Hardik|27|Bhopal|8500.00|

|6|Komal|22| MP       |4500.00|

|7|Muffy|24|Indore|10000.00|

+----+----------+-----+-----------+----------+

7 rows inset(0.00 sec)
```

The following table is a simple example showing the usage of various SQL Boolean Expressions −

```
SQL> SELECT * FROM CUSTOMERS WHERE SALARY =10000;

+----+-------+-----+---------+----------+
| ID | NAME  | AGE | ADDRESS | SALARY   |

+----+-------+-----+---------+----------+

|7|Muffy|24|Indore|10000.00|

+----+-------+-----+---------+----------+

1 row inset(0.00 sec)
```

2. Numeric Expression

These expressions are used to perform any mathematical operation in any query. Following is the syntax –

```
SELECT numerical_expression as  OPERATION_NAME
[FROM table_name
WHERE CONDITION] ;
```

Here, the numerical_expression is used for a mathematical expression or any formula. Following is a simple example showing the usage of SQL Numeric Expressions –

```
SQL> SELECT (15+6) AS ADDITION

+----------+

| ADDITION |

+----------+

|21|

+----------+

1 row inset(0.00 sec)
```

There are several built-in functions like avg(), sum(), count(), etc., to perform what is known as the aggregate data calculations against a table or a specific table column.

```
SQL> SELECT COUNT(*) AS "RECORDS" FROM CUSTOMERS;

+---------+

| RECORDS |
+---------+

|7|

+---------+

1 row inset(0.00 sec)
```

3. Date Expressions

Date Expressions return current system date and time values −

```
SQL>  SELECT CURRENT_TIMESTAMP;

+---------------------+

|Current_Timestamp|

+---------------------+

|2009-11-1206:40:23|

+---------------------+

1 row inset(0.00 sec)
```

Another date expression is as shown below −

```
SQL>  SELECT  GETDATE();;

+-------------------------+

| GETDATE               |

+-------------------------+

|2009-10-2212:07:18.140|

+-------------------------+

1 row inset(0.00 sec)
```

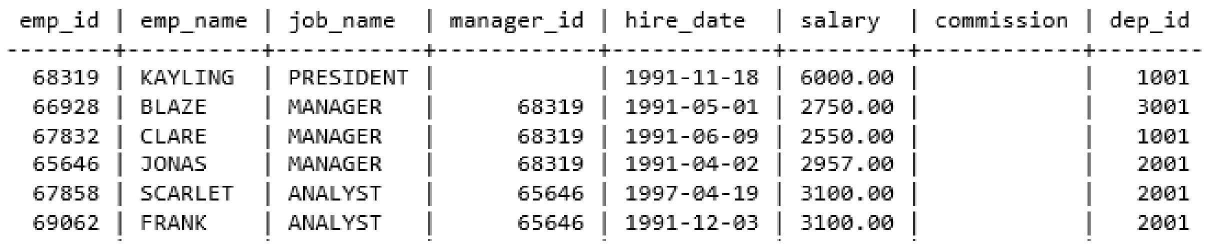
Lab Activity:

Students kindly perform the above discussed topics in your lab sessions.

Lab Activity :

Students kindly implement the below questions in your lab sessions based on the above studied concepts.

1. Write a query in SQL to display a table containing employee details as shown in the picture below:



```
 emp_id | emp_name | job_name  | manager_id | hire_date  | salary  | commission | dep_id
--------+----------+-----------+------------+------------+---------+------------+--------
  68319 | KAYLING  | PRESIDENT |            | 1991-11-18 | 6000.00 |            |   1001
  66928 | BLAZE    | MANAGER   |      68319 | 1991-05-01 | 2750.00 |            |   3001
  67832 | CLARE    | MANAGER   |      68319 | 1991-06-09 | 2550.00 |            |   1001
  65646 | JONAS    | MANAGER   |      68319 | 1991-04-02 | 2957.00 |            |   2001
  67858 | SCARLET  | ANALYST   |      65646 | 1997-04-19 | 3100.00 |            |   2001
  69062 | FRANK    | ANALYST   |      65646 | 1991-12-03 | 3100.00 |            |   2001
```

Here, students use CREATE and INSERT command to create the table and insert the values.

2. Use SELECT command to display the table created above.

3. Use ALTER and then DROP command to drop the column “commission”.

3. Use ALTER command and then ADD command to add column “commission” back.

4. Use UPDATE column to add some values to the column “commission” where dep_id=1001 OR dep_id=3001.

5. Use DELETE command to delete rows where job_name=”ANALYST”.

6. Use DELETE command to delete rows where job_name=”MANAGER” and emp_name=”CLARE”.

6. Use UPDATE command to add manager_id as 68310 where job_name=”PRESIDENT”.

7. Use TRUNCATE command to remove all the rows from the table. And then use SELECT command to check whether the table contents exist or not.

8. Use DROP command to drop the table.

ORDER BY, GROUP BY, HAVING Clauses

1. SQL - ORDER BY clause

The ORDER BY keyword is used to sort the result-set in ascending or descending order.The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

A query with its various clauses (FROM, WHERE, GROUP BY, HAVING) determines the rows to be selected and the columns. The order of rows is not fixed unless an ORDER BY clause is given.

The columns to be used for ordering are specified by using the "column names" or by specifying the "serial number" of the column in the SELECT list. • The sort is done on the column in "ascending" or "descending" order. By default the ordering of data is "ascending" order.

Syntax

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

ORDER BY Example

Consider the CUSTOMERS table having the following records −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
```

The following code block has an example, which would sort the result in an ascending order by the NAME and the SALARY −

```
SQL> SELECT * FROM CUSTOMERS

  ORDER BY NAME, SALARY;
```

This would produce the following result −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
+----+----------+-----+-----------+----------+
```

The following code block has an example, which would sort the result in the descending order by NAME.

```
SQL> SELECT * FROM CUSTOMERS

  ORDER BY NAME DESC;
```

This would produce the following result –

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
+----+----------+-----+-----------+----------+
```

2. SQL - GROUP BY clause

The SQL GROUP BY clause is used in collaboration with the SELECT statement to arrange identical data into groups. This GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

Syntax

The basic syntax of a GROUP BY clause is shown in the following code block. The GROUP BY clause must follow the conditions in the WHERE clause and must precede the ORDER BY clause if one is used.

```
SELECT column1, column2
FROM table_name
WHERE [ conditions ]
GROUP BY column1, column2
ORDER BY column1, column2
```

Example

Consider the CUSTOMERS table is having the following records –

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

If you want to know the total amount of the salary on each customer, then the GROUP BY query would be as follows.

```
SQL> SELECT NAME, SUM(SALARY) FROM CUSTOMERS

   GROUP BY NAME;
```

This would produce the following result −

```
+----------+-------------+
| NAME     | SUM(SALARY) |
+----------+-------------+
| Chaitali |     6500.00 |
| Hardik   |     8500.00 |
| kaushik  |     2000.00 |
| Khilan   |     1500.00 |
| Komal    |     4500.00 |
| Muffy    |    10000.00 |
| Ramesh   |     2000.00 |
+----------+-------------+
```

3. SQL - Having Clause

The HAVING Clause enables you to specify conditions that filter which group results appear in the results.

The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause.

Syntax

The following code block shows the position of the HAVING Clause in a query.

```
SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY
```

Example

Consider the CUSTOMERS table having the following records.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Following is an example, which would display a record for a similar age count that would be more than or equal to 2.

```
SQL > SELECT ID, NAME, AGE, ADDRESS, SALARY

FROM CUSTOMERS

GROUP BY age

HAVING COUNT(age) >= 2;
```

This would produce the following result −

```
+----+--------+-----+---------+---------+
| ID | NAME   | AGE | ADDRESS | SALARY  |
+----+--------+-----+---------+---------+
|  2 | Khilan |  25 | Delhi   | 1500.00 |
```

Wild Cards

The SQL LIKE clause is used to compare a value to similar values using wildcard operators. There are two wildcards used in conjunction with the LIKE operator.

- **The percent sign (%)**
- **The underscore (_)**

The percent sign represents zero, one or multiple characters. The underscore represents a single number or character.

Syntax

The basic syntax of % and _ is as follows −

```
SELECT FROM table_name
WHERE column LIKE 'XXXX%'

or
```

```
SELECT FROM table_name
WHERE column LIKE '%XXXX%'

or

SELECT FROM table_name
WHERE column LIKE 'XXXX_'

or

SELECT FROM table_name
WHERE column LIKE '_XXXX'

or

SELECT FROM table_name
WHERE column LIKE '_XXXX_'
```

Example

The following table has a few examples showing the WHERE part having different LIKE clause with '%' and '_' operators –

Sr.No.	Statement & Description
1	WHERE SALARY LIKE '200%' Finds any values that start with 200.
2	WHERE SALARY LIKE '%200%' Finds any values that have 200 in any position.
3	WHERE SALARY LIKE '_00%' Finds any values that have 00 in the second and third positions.

4	WHERE SALARY LIKE '2_%_%' Finds any values that start with 2 and are at least 3 characters in length.
5	WHERE SALARY LIKE '%2' Finds any values that end with 2.
6	WHERE SALARY LIKE '_2%3' Finds any values that have a 2 in the second position and end with a 3.
7	WHERE SALARY LIKE '2___3' Finds any values in a five-digit number that start with 2 and end with 3.

Following is an example, which would display all the records from the CUSTOMERS table, where the SALARY starts with 200.

```
SQL> SELECT * FROM CUSTOMERS

WHERE SALARY LIKE '200%';
```

This would produce the following result −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
+----+----------+-----+-----------+----------+
```

Lab Activity:

Students kindly perform the above discussed topics i.e. Order by, Group by, Having Clauses and Wild cards in your lab sessions and also the below given questions.

Lab Activity:

A. Create table “customers” as shown below:

CustomerID	**CustomerName**	**City**	**PostalCode**	**Country**
1	Alfred	Berlin	12209	Germany
2	Ana	México	05021	Mexico
3	Antonio	México	05023	Mexico
4	Harry	London	WA1 1DP	UK
5	Sierra	Luleå	S-958 22	Sweden

Perform the below functions on the above table:

1.Selects all customers from the "Customers" table, sorted by the "Country" column.Hint:

```
SELECT * FROM Customers
ORDER BY Country;
```

2. Write SQL statement lists the number of customers in each country.Hint:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country;
```

3. The following SQL statement lists the number of customers in each country. Only include countries with more than 5 customers.Hint:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5;
```

4.The following SQL statement lists the number of customers in each country, sorted high to low (Only include countries with more than 5 customers).

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5
ORDER BY COUNT(CustomerID) DESC;
```

Constraints

SQL Constraints

Constraints are the rules enforced on data columns on a table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

Following are some of the most commonly used constraints available in SQL –

- **NOT NULL** - Ensures that a column cannot have a NULL value
- **UNIQUE** - Ensures that all values in a column are different
- **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- **FOREIGN KEY** - Uniquely identifies a row/record in another table
- **CHECK** - Ensures that all values in a column satisfies a specific condition
- **DEFAULT** - Sets a default value for a column when no value is specified
- **INDEX** - Used to create and retrieve data from the database very quickly

1. NOT NULL Constraint – Ensures that a column cannot have a NULL value.

The NOT NULL constraint enforces a column to NOT accept NULL values.

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

For example, the following SQL query creates a new table called CUSTOMERS and adds five columns, three of which, are ID NAME and AGE, In this we specify not to accept NULLs –

1. SQL NOT NULL on CREATE TABLE

```
CREATE TABLE CUSTOMERS(

   ID   INT            NOT NULL,

   NAME VARCHAR (20)     NOT NULL,

   AGE  INT            NOT NULL,

   ADDRESS  CHAR (25) ,

   SALARY   DECIMAL (18, 2),

   PRIMARY KEY (ID)

);
```

2. SQL NOT NULL on ALTER TABLE

To create a NOT NULL constraint on the "Salary" column when the "Customers" table is already created, use the following SQL:

```
ALTER TABLE Persons
MODIFY Age int NOT NULL;
```

2. DEFAULT Constraint − Provides a default value for a column when none is specified.

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

For example, the following SQL creates a new table called CUSTOMERS and adds five columns. Here, the SALARY column is set to 5000.00 by default, so in case the INSERT INTO statement does not provide a value for this column, then by default this column would be set to 5000.00.

```
CREATE TABLE CUSTOMERS(
  ID   INT            NOT NULL,

  NAME VARCHAR (20)     NOT NULL,

  AGE  INT            NOT NULL,

  ADDRESS  CHAR (25) ,

  SALARY   DECIMAL (18, 2) DEFAULT 5000.00,

  PRIMARY KEY (ID)

);
```

3. UNIQUE Constraint − Ensures that all the values in a column are different.

Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.

A PRIMARY KEY constraint automatically has a UNIQUE constraint.

However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

For example, the following SQL query creates a new table called CUSTOMERS and adds five columns. Here, the AGE column is set to UNIQUE, so that you cannot have two records with the same age.

```
CREATE TABLE CUSTOMERS(

  ID   INT            NOT NULL,

  NAME VARCHAR (20)     NOT NULL,

  AGE  INT            NOT NULL UNIQUE,

  ADDRESS  CHAR (25) ,

  SALARY   DECIMAL (18, 2),

PRIMARY KEY (ID)

);
```

4. PRIMARY Key Constraint− Uniquely identifies each row/record in a database table.

Primary keys must contain UNIQUE values, and cannot contain NULL values.

A table can have only one primary key, which may consist of single or multiple fields.

Create Primary Key

Here is the syntax to define the ID attribute as a primary key in a CUSTOMERS table.

```
CREATE TABLE CUSTOMERS(
   ID   INT              NOT NULL,
   NAME VARCHAR (20)     NOT NULL,
   AGE  INT              NOT NULL,
   ADDRESS  CHAR (25) ,
   SALARY   DECIMAL (18, 2),
   PRIMARY KEY (ID)
);
```

To create a PRIMARY KEY constraint on the "ID" column when the CUSTOMERS table already exists, use the following SQL syntax −

```
ALTER TABLE CUSTOMER ADD PRIMARY KEY (ID);
```

5. FOREIGN Key Constraint − Uniquely identifies a row/record in any another database table.

A foreign key is a key used to link two tables together. This is sometimes also called as a referencing key.

A Foreign Key is a column or a combination of columns whose values match a Primary Key in a different table.

The relationship between 2 tables matches the Primary Key in one of the tables with a Foreign Key in the second table.

Example

Consider the structure of the following two tables.

CUSTOMERS table

```
CREATE TABLE CUSTOMERS(

   ID   INT              NOT NULL,

   NAME VARCHAR (20)     NOT NULL,
   AGE  INT            NOT NULL,

   ADDRESS  CHAR (25) ,

   SALARY   DECIMAL (18, 2),

   PRIMARY KEY (ID)

);
```

ORDERS table

```
CREATE TABLE ORDERS (

  ID        INT       NOT NULL,

  DATE       DATETIME,

  CUSTOMER_ID INT references CUSTOMERS(ID),

  AMOUNT     double,

  PRIMARY KEY (ID)

);
```

If the ORDERS table has already been created and the foreign key has not yet been set, the use the syntax for specifying a foreign key by altering a table.

```
ALTER TABLE ORDERS
  ADD FOREIGN KEY (Customer_ID) REFERENCES CUSTOMERS (ID);
```

6. CHECK Constraint − The CHECK constraint ensures that all values in a column satisfy certain conditions.

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.

For example, the following program creates a new table called CUSTOMERS and adds five columns. Here, we add a CHECK with AGE column, so that you cannot have any CUSTOMER who is below 18 years.

```
CREATE TABLE CUSTOMERS(
   ID   INT            NOT NULL,
   NAME VARCHAR (20)     NOT NULL,
   AGE  INT            NOT NULL CHECK (AGE >= 18),
   ADDRESS  CHAR (25) ,
   SALARY   DECIMAL (18, 2),
   PRIMARY KEY (ID)
);
```

If the CUSTOMERS table has already been created, then to add a CHECK constraint to AGE column, you would write a statement like the one given below.

```
ALTER TABLE CUSTOMERS
   MODIFY AGE INT NOT NULL CHECK (AGE >= 18 );
```

7. INDEX − Used to create and retrieve data from the database very quickly. An Index can be created by using a single or group of columns in a table.

Indexes are used to retrieve data from the database very fast. The users cannot see the indexes, they are just used to speed up searches/queries.

Tip: Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So, only create indexes on columns that will be frequently searched against.

For example, the following SQL syntax creates a new table called CUSTOMERS and adds five columns in it.

```
CREATE TABLE CUSTOMERS(
   ID   INT            NOT NULL,
   NAME VARCHAR (20)     NOT NULL,
   AGE  INT            NOT NULL,
   ADDRESS  CHAR (25) ,

   SALARY   DECIMAL (18, 2),

   PRIMARY KEY (ID)

);
```

Now, you can create an index on a single or multiple columns using the syntax given below.

```
CREATE INDEX index_name
  ON table_name ( column1, column2.....);
```

To create an INDEX on the AGE column, to optimize the search on customers for a specific age, you can use the follow SQL syntax which is given below −

```
CREATE INDEX idx_age
  ON CUSTOMERS ( AGE );
```

Students kindly perform the above discussed topics in your lab sessions.

Joins

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.Different Types of SQL JOINs are:

- **(INNER) JOIN**: Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN**: Return all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN**: Return all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN**: Return all records when there is a match in either left or right table

Example: Consider the two tables below:

Student

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	HARSH	DELHI	8759770477	18
2	PRATIK	BIHAR	7333834034	19
3	RIYANKA	SILIGURI	9876543210	20
4	DEEP	RAMNAGAR	8520369741	18
5	SAPTARHI	KOLKATA	9654783210	19
6	DHANRAJ	BARABAJAR	7412589630	20
7	ROHIT	BALURGHAT	9630258741	18
8	NIRAJ	ALIPUR	7412356890	19

StudentCourse

COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4
1	5
4	9
5	10
4	11

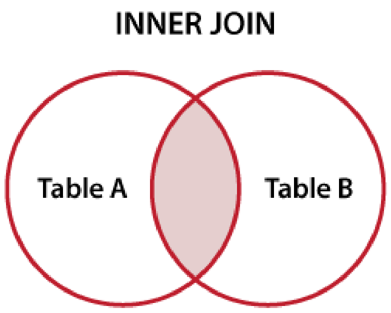
1. **INNER JOIN**: The INNER JOIN keyword selects all rows from both the tables as long as the condition satisfies. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be same.

   Syntax:

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
INNER JOIN table2
ON table1.matching_column = table2.matching_column;


table1: First table.
table2: Second table
```

```
matching_column: Column common to both the tables.
```

Tip: We can also write JOIN instead of INNER JOIN. JOIN is same as INNER JOIN.



Example Queries(INNER JOIN)

This query will show the names and age of students enrolled in different courses.

```
SELECT StudentCourse.COURSE_ID, Student.NAME, Student.AGE FROM Student
INNER JOIN StudentCourse
ON Student.ROLL_NO = StudentCourse.ROLL_NO;
```

Output:

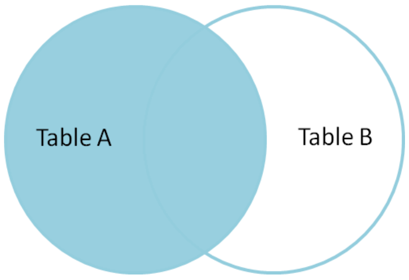
COURSE_ID	NAME	Age
1	HARSH	18
2	PRATIK	19
2	RIYANKA	20
3	DEEP	18
1	SAPTARHI	19

2. **LEFT JOIN**: This join returns all the rows of the table on the left side of the join and matching rows for the table on the right side of join. The rows for which there is no matching row on right side, the result-set will contain null. LEFT JOIN is also known as LEFT OUTER JOIN.
Syntax:

```
SELECT table1.column1,table1.column2,table2.column1,....
```

```
FROM table1
LEFT JOIN table2
ON table1.matching_column = table2.matching_column;
table1: First table.
table2: Second table
matching_column: Column common to both the tables.
```

Tip: We can also use LEFT OUTER JOIN instead of LEFT JOIN, both are same.



Example Queries(LEFT JOIN):

```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
LEFT JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output:

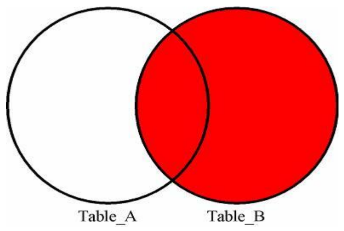
NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	*NULL*
ROHIT	*NULL*
NIRAJ	*NULL*

3. **RIGHT JOIN**: RIGHT JOIN is similar to LEFT JOIN. This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of join. The rows for which there is no matching row on left side, the result-set will contain null. RIGHT JOIN is also known as RIGHT OUTER JOIN.

   Syntax:

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
RIGHT JOIN table2
ON table1.matching_column = table2.matching_column;


table1: First table.
table2: Second table
matching_column: Column common to both the tables.
```

Tip: We can also use RIGHT OUTER JOIN instead of RIGHT JOIN, both are same.



Example Queries(RIGHT JOIN):

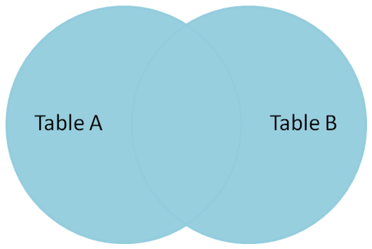
```
SELECT Student.NAME,StudentCourse.COURSE_ID

FROM Student

RIGHT JOIN StudentCourse

ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output:

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
NULL	4
NULL	5
NULL	4

4. **FULL JOIN:** FULL JOIN creates the result-set by combining result of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both the tables. The rows for which there is no matching, the result-set will contain NULL values.
   Syntax:

```
SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

FULL JOIN table2

ON table1.matching_column = table2.matching_column;



table1: First table.

table2: Second table

matching_column: Column common to both the tables.
```



Example Queries(FULL JOIN):

```
SELECT Student.NAME,StudentCourse.COURSE_ID

FROM Student

FULL JOIN StudentCourse

ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output:

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	*NULL*
ROHIT	*NULL*
NIRAJ	*NULL*
NULL	9
NULL	10
NULL	11

SQL | Join (Cartesian Join & Self Join)

- CARTESIAN JOIN
- SELF JOIN
- Equi Join
- Non-Equi Join

Consider the two tables below:

Student

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	Delhi	9156768971	18

StudentCourse

COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4

1. **CARTESIAN JOIN:**

The CARTESIAN JOIN is also known as CROSS JOIN. In a CARTESIAN JOIN there is a join for each row of one table to every row of another table. This usually happens when the matching column or WHERE condition is not specified.

- In the absence of a WHERE condition the CARTESIAN JOIN will behave like a CARTESIAN PRODUCT . i.e., the number of rows in the result-set is the product of the number of rows of the two tables.
- In the presence of WHERE condition this JOIN will function like a INNER JOIN.
- Generally speaking, Cross join is similar to an inner join where the join-condition will always evaluate to True

Syntax:

```
SELECT table1.column1 , table1.column2, table2.column1...

FROM table1

CROSS JOIN table2;




table1: First table.
table2: Second table
```

Example Queries(CARTESIAN JOIN):
In the below query we will select NAME and Age from Student table and COURSE_ID from StudentCourse table. In the output you can see that each row of the table Student is joined with every row of the table StudentCourse. The total rows in the result-set = 4 * 4 = 16.

```
SELECT Student.NAME, Student.AGE, StudentCourse.COURSE_ID

FROM Student

CROSS JOIN StudentCourse;
```

Output:

NAME	AGE	COURSE_ID
Ram	18	1
Ram	18	2
Ram	18	2
Ram	18	3
RAMESH	18	1
RAMESH	18	2
RAMESH	18	2
RAMESH	18	3
SUJIT	20	1
SUJIT	20	2
SUJIT	20	2
SUJIT	20	3
SURESH	18	1
SURESH	18	2
SURESH	18	2
SURESH	18	3

2. **SELF JOIN**:

As the name signifies, in SELF JOIN a table is joined to itself. That is, each row of the table is joined with itself and all other rows depending on some conditions. In other words we can say that it is a join between two copies of the same table.

Syntax:

```
SELECT a.coulmn1 , b.column2

FROM table_name a, table_name b

WHERE some_condition;


table_name: Name of the table.
some_condition: Condition for selecting the rows.
```

Sometimes is required to join the table to itself. To join a table to itself, “two copies” of the same table have to be opened in the memory.

- Since the table names are the same, the second table will overwrite the first table. In effect, this will result in only one table being in memory.
  - ➢ This is because a table name is translated into a specific memory location.
- Hence in the FROM clause, the table name needs to be mentioned twice with an “alias”
  - ➢ These two table aliases will cause two identical tables to be opened in different memory locations.
  - ➢ This will result in two identical tables to be physically present in the computer memory.

Example Queries(SELF JOIN):

```
SELECT a.ROLL_NO , b.NAME

FROM Student a, Student b

WHERE a.ROLL_NO < b.ROLL_NO;
```

Output:

ROLL_NO	NAME
1	RAMESH
1	SUJIT
2	SUJIT
1	SURESH
2	SURESH
3	SURESH

3. EQUI JOIN

In an Equijoin, the WHERE statement compares two columns from two tables with the equivalence operator "=". This JOIN returns all rows from both tables, where there is a match.

Syntax:

```
SELECT <col1>, <col2>,…

FROM <table1>,<table2>

Where <table1>.<col1>=<table2>.<col2>

[AND <condition>] [ORDER BY <col1>, <col2>,…]
```

- Equi Join which is sometimes also referred to as Inner Join or simple join is done by writing a join condition using the "=" operator
- Typically the tables are joint to get meaningful data.
- The join is based on the equality of column values in the two tables and therefore is called an Equijoin.
- To join together "n" tables, you need a minimum of "n-1" JOIN conditions.

For example: To join three tables, a minimum of two joins is required.

In the syntax given in the slide:

Column1 in Table1 is usually the Primary key of that table.

Column2 in Table2 is a Foreign key in that table.

Column1 and Column2 must have the same data type, and for certain data types, they should have same size, as well

Example1 Equi Join:

To display student code and name along with thedepartment name to which they belong

```
SELECT Student_Code,Student_name,Dept_name

FROM Student_Master ,Department_Master

WHERE Student_Master.Dept_code =Department_Master.Dept_code;
```

Example 2:

To display student and staff name along with thedepartment name to which they belong

```
SELECT student_name,staff_name, dept_name

FROM student_master, department_master,staff_master

WHERE student_master.dept_code=department_master.dept_code and

staff_master.dept_code=department_master.dept_code;
```

4. NON-EQUI JOIN

A non-equi join is based on condition other than an equality operator

Example: To display details of staff_members who receive salary in the range defined as per grade

```
SELECT s.staff_name,s.staff_sal,sl.grade
FROM staff_master s,salgrade sl
WHERE staff_sal BETWEEN sl.losal and sl.hisal
```

A non-equi join is based on condition other than an equality operator

Example: To display details of staff_members who receive salary in the range defined as per grade

```
SELECT s.staff_name,s.staff_sal,sl.grade

FROM staff_master s,salgrade sl

WHERE staff_sal BETWEEN sl.losal and sl.hisal
```

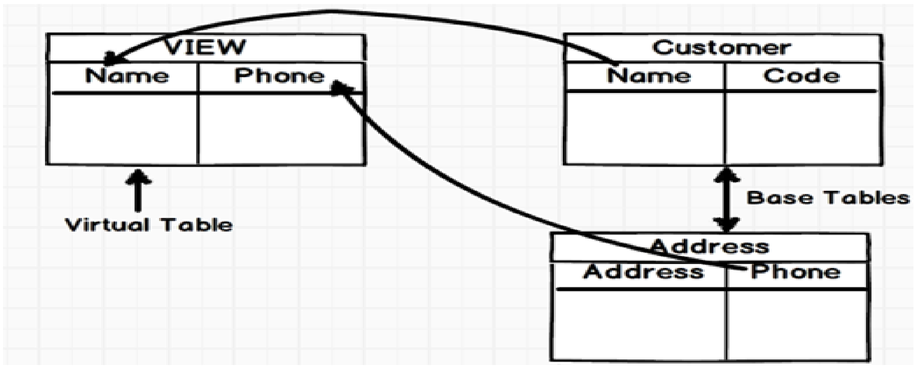
Name	Type
GRADE	NUMBER
LOSAL	NUMBER
HISAL	NUMBER

So to display all the staff members who receive salary between the ranges specified in the salgrade table we will use a non-equijoin.

Lab Activity:
Students kindly perform the above discussed topics in your lab sessions.

Views

SQL | Views



- Views in SQL are kind of virtual tables.
- A view also has rows and columns as they are in a real table in the database.
- We can create a view by selecting fields from one or more tables present in the database.
- The VIEW can be treated as a base table and it can be QUERIED, UPDATED, INSERTED INTO, DELETED FROM and JOINED with other tables and views.
- A VIEW is a data object which does not contain any data. Its contents are the resultant of a base table. They are operated just like the base table but they don't contain any data of their own.
- A view can be accessed with the use of SQL SELECT statement like a table. A view can also be made up by selecting data from more than one tables.

We can create View using CREATE VIEW statement. A View can be created from a single table or multiple tables.
Syntax:

```
CREATE VIEW view_name AS

SELECT column1, column2.....

FROM table_name

WHERE condition;

view_name: Name for the View

table_name: Name of the table
condition: Condition to select rows
```

Examples:

Consider the CUSTOMERS table having the following records –

```
+----+----------+-----+-----------+----------+

| ID | NAME     | AGE | ADDRESS   | SALARY   |

+----+----------+-----+-----------+----------+

|1|Ramesh|32|Ahmedabad|2000.00|

|2|Khilan|25|Delhi|1500.00|

|3| kaushik  |23|Kota|2000.00|

|4|Chaitali|25|Mumbai|6500.00|

|5|Hardik|27|Bhopal|8500.00|

|6|Komal|22| MP      |4500.00|
|7|Muffy|24|Indore|10000.00|

+----+----------+-----+-----------+----------+
```

1. Following is an example **to create a view** from the CUSTOMERS table. This view would be used to have customer name and age from the CUSTOMERS table.

```
SQL > CREATE VIEW CUSTOMERS_VIEW AS

SELECT name, age

FROM  CUSTOMERS;
```

Now, you can query CUSTOMERS_VIEW in a similar way as you query an actual table. Following is an example for the same.

```
SQL > SELECT * FROM CUSTOMERS_VIEW;
```

This would produce the following result.

```
+----------+-----+
| name     | age |
+----------+-----+
| Ramesh   |  32 |
| Khilan   |  25 |
| kaushik  |  23 |
| Chaitali |  25 |
| Hardik   |  27 |
| Komal    |  22 |
| Muffy    |  24 |
+----------+-----+
```

2. The WITH CHECK OPTION

The WITH CHECK OPTION is a CREATE VIEW statement option. The purpose of the WITH CHECK OPTION is to ensure that all UPDATE and INSERTs satisfy the condition(s) in the view definition.

If they do not satisfy the condition(s), the UPDATE or INSERT returns an error.

The following code block has an example of creating same view CUSTOMERS_VIEW with the WITH CHECK OPTION.

```
CREATE VIEW CUSTOMERS_VIEW AS

SELECT name, age
```

```
FROM  CUSTOMERS

WHERE age IS NOT NULL

WITH CHECK OPTION;
```

The WITH CHECK OPTION in this case should deny the entry of any NULL values in the view's AGE column, because the view is defined by data that does not have a NULL value in the AGE column.

3. Updating a View

A view can be updated under certain conditions which are given below −

- The SELECT clause may not contain the keyword DISTINCT.
- The SELECT clause may not contain summary functions.
- The SELECT clause may not contain set functions.
- The SELECT clause may not contain set operators.
- The SELECT clause may not contain an ORDER BY clause.
- The FROM clause may not contain multiple tables.
- The WHERE clause may not contain subqueries.
- The query may not contain GROUP BY or HAVING.
- Calculated columns may not be updated.
- All NOT NULL columns from the base table must be included in the view in order for the INSERT query to function.

So, if a view satisfies all the above-mentioned rules then you can update that view. The following code block has an example to update the age of Ramesh.

```
SQL > UPDATE CUSTOMERS_VIEW

  SET AGE =35

  WHERE name ='Ramesh';
```

This would ultimately update the base table CUSTOMERS and the same would reflect in the view itself. Now, try to query the base table and the SELECT statement would produce the following result.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  35 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

4. Inserting Rows into a View

Rows of data can be inserted into a view. The same rules that apply to the UPDATE command also apply to the INSERT command.

Here, we cannot insert rows in the CUSTOMERS_VIEW because we have not included all the NOT NULL columns in this view, otherwise you can insert rows in a view in a similar way as you insert them in a table.

5. Deleting Rows into a View

Rows of data can be deleted from a view. The same rules that apply to the UPDATE and INSERT commands apply to the DELETE command.

Following is an example to delete a record having AGE = 22.

```
SQL > DELETE FROM CUSTOMERS_VIEW

  WHERE age =22;
```

This would ultimately delete a row from the base table CUSTOMERS and the same would reflect in the view itself. Now, try to query the base table and the SELECT statement would produce the following result.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  35 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

6. Dropping Views

Obviously, where you have a view, you need a way to drop the view if it is no longer needed. The syntax is very simple and is given below −

```
DROP VIEW view_name;
```

Following is an example to drop the CUSTOMERS_VIEW from the CUSTOMERS table.

```
DROP VIEW CUSTOMERS_VIEW;
```

Lab Activity:

Students kindly perform the above discussed topics in your lab sessions.

Functions

What are SQL Functions?

SQL provides many built-in functions to perform operations on data. These functions are useful while performing mathematical calculations, string concatenations, sub-strings etc. SQL functions are divided into two categories,

1. Aggregate Functions
2. Scalar Functions

Aggregate Functions

These functions return a single value after performing calculations on a group of values. Following are some of the frequently used Aggregrate functions.

1. AVG() Function:

Average returns average value after calculating it from values in a numeric column.

Its general syntax is,

```
SELECT AVG(column_name) FROM table_name
```

Using AVG() function

Consider the following Emp table

eid	name	age	salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

SQL query to find average salary will be,

```
SELECT avg(salary) from Emp;
```

Result of the above query will be,

avg(salary)
8200

2. COUNT() Function

Count returns the number of rows present in the table either based on some condition or without condition.

Its general syntax is,

```
SELECT COUNT(column_name) FROM table-name
```

Using COUNT() function:

Consider the following Emp table

eid	name	age	salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

SQL query to count employees, satisfying specified condition is,

```
SELECT COUNT(name) FROM Emp WHERE salary = 8000;
```

Result of the above query will be,

count(name)
2

Example of COUNT(distinct):

Consider the following Emp table

eid	name	age	salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

SQL query is,

```
SELECT COUNT(DISTINCT salary) FROM emp;
```

Result of the above query will be,

count(distinct salary)
4

3. FIRST() Function:

First function returns first value of a selected column

Syntax for FIRST function is,

```
SELECT FIRST(column_name) FROM table-name;
```

Using FIRST() function:

Consider the following Emp table

Eid	**name**	**age**	**salary**
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

SQL query will be,

```
SELECT FIRST(salary) FROM Emp;
```

and the result will be,

first(salary)
9000

4. LAST() Function

LAST function returns the return last value of the selected column.

Syntax of LAST function is,

```
SELECT LAST(column_name) FROM table-name;
```

Using LAST() function

Consider the following Emp table

Eid	name	age	salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

SQL query will be,

```
SELECT LAST(salary) FROM emp;
```

Result of the above query will be,

last(salary)
8000

5. MAX() Function

MAX function returns maximum value from selected column of the table.

Syntax of MAX function is,

```
SELECT MAX(column_name) from table-name;
```

Using MAX() function

Consider the following Emp table

Eid	**name**	**age**	**salary**
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

SQL query to find the Maximum salary will be,

```
SELECT MAX(salary) FROM emp;
```

Result of the above query will be,

MAX(salary)
10000

6. MIN() Function

MIN function returns minimum value from a selected column of the table.

Syntax for MIN function is,

```
SELECT MIN(column_name) from table-name;
```

Using MIN() function

Consider the following Emp table,

Eid	name	age	salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

SQL query to find minimum salary is,

```
SELECT MIN(salary) FROM emp;
```

Result will be,

MIN(salary)

```
6000
```

7. SUM() Function

SUM function returns total sum of a selected columns numeric values.

Syntax for SUM is,

```
SELECT SUM(column_name) from table-name;
```

Using SUM() function:

Consider the following Emp table

Eid	name	age	salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

SQL query to find sum of salaries will be,

```
SELECT SUM(salary) FROM emp;
```

Result of above query is,

SUM(salary)
41000

Scalar Functions

Scalar functions return a single value from an input value. Following are some frequently used Scalar Functions in SQL.

1. **UCASE() Function**

UCASE function is used to convert value of string column to Uppercase characters.

Syntax of UCASE,

```
SELECT UCASE(column_name) from table-name;
```

Using UCASE() function

Consider the following Emp table

Eid	name	age	salary
401	anu	22	9000
402	shane	29	8000
403	rohan	34	6000
404	scott	44	10000
405	Tiger	35	8000

SQL query for using UCASE is,

```
SELECT UCASE(name) FROM emp;
```

Result is,

UCASE(name)
ANU
SHANE
ROHAN
SCOTT
TIGER

2. LCASE() Function

LCASE function is used to convert value of string columns to Lowecase characters.

Syntax for LCASE is,

```
SELECT LCASE(column_name) FROM table-name;
```

Using LCASE() function

Consider the following Emp table

Eid	name	age	salary
401	Anu	22	9000

402	Shane	29	8000
403	Rohan	34	6000
404	SCOTT	44	10000
405	Tiger	35	8000

SQL query for converting string value to Lower case is,

```
SELECT LCASE(name) FROM emp;
```

Result will be,

LCASE(name)
Anu
Shane
Rohan
Scott
Tiger

3. MID() Function

MID function is used to extract substrings from column values of string type in a table.

Syntax for MID function is,

```
SELECT MID(column_name, start, length) from table-name;
```

Using MID() function

Consider the following Emp table

Eid	name	age	salary
401	anu	22	9000
402	shane	29	8000
403	rohan	34	6000
404	scott	44	10000
405	Tiger	35	8000

SQL query will be,

```
SELECT MID(name,2,2) FROM emp;
```

Result will come out to be,

MID(name,2,2)

Nu
Ha
Oh
Co
Ig

4. ROUND() Function

ROUND function is used to round a numeric field to number of nearest integer. It is used on Decimal point values.

Syntax of Round function is,

```
SELECT ROUND(column_name, decimals) from table-name;
```

Using ROUND() function

Consider the following Emp table

eid	name	age	salary
401	anu	22	9000.67
402	shane	29	8000.98
403	rohan	34	6000.45

404	scott	44	10000
405	Tiger	35	8000.01

SQL query is,

```
SELECT ROUND(salary) from emp;
```

Result will be,

ROUND(salary)
9001
8001
6000
10000
8000

Lab Activity:
Students kindly perform the above discussed topics in your lab sessions.

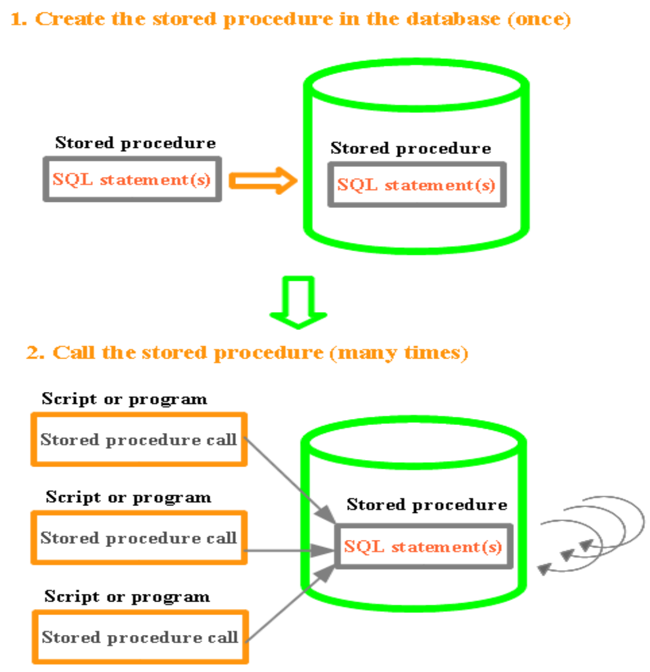
Stored Procedure

What is a Stored Procedure?

A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.

So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.

You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.



Stored Procedure Syntax:

```
CREATE PROCEDURE procedure_name
AS
sql_statement
GO;
```

Execute a Stored Procedure:

```
EXEC procedure_name;
```

Demo Database:

Below is a selection from the "Customers" table in the Northwind sample database:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfred	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

Stored Procedure Example

The following SQL statement creates a stored procedure named "SelectAllCustomers" that selects all records from the "Customers" table:

Example

```
CREATE PROCEDURE SelectAllCustomers
AS
SELECT * FROM Customers
GO;
```

Execute the stored procedure above as follows:

Example

```
EXEC SelectAllCustomers;
```

Stored Procedure With One Parameter

The following SQL statement creates a stored procedure that selects Customers from a particular City from the "Customers" table:

Example

```
CREATE PROCEDURE SelectAllCustomers @City nvarchar(30)
AS
SELECT * FROM Customers WHERE City = @City
GO;
```

Execute the stored procedure above as follows:

Example

```
EXEC SelectAllCustomers City = "London";
```

Stored Procedure With Multiple Parameters

Setting up multiple parameters is very easy. Just list each parameter and the data type separated by a comma as shown below.

The following SQL statement creates a stored procedure that selects Customers from a particular City with a particular PostalCode from the "Customers" table:

Example

```
CREATE PROCEDURE SelectAllCustomers @City nvarchar(30), @PostalCode nvarchar(10)
AS
SELECT * FROM Customers WHERE City = @City AND PostalCode = @PostalCode
GO;
```

Execute the stored procedure above as follows:

Example

```
EXEC SelectAllCustomers City = "London", PostalCode = "WA1 1DP";
```

Lab Activity:

Students kindly perform the above discussed topics in your lab sessions.

SQL : UNION and UNION ALL Clauses

UNION Clause:

The SQL UNION clause/operator is used to combine the results of two or more SELECT statements without returning any duplicate rows.

To use this UNION clause, each SELECT statement must have

- The same number of columns selected
- The same number of column expressions
- The same data type and
- Have them in the same order, but they need not have to be in the same length.

Syntax

The basic syntax of a UNION clause is as follows −

```
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]

UNION

SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
```

Here, the given condition could be any given expression based on your requirement.

Example

Consider the following two tables.

Table 1 − CUSTOMERS Table is as follows.

```
+----+----------+-----+-----------+----------+

| ID | NAME     | AGE | ADDRESS   | SALARY   |

+----+----------+-----+-----------+----------+

|1|Ramesh|32|Ahmedabad|2000.00|

|2|Khilan|25|Delhi|1500.00|

|3| kaushik  |23|Kota|2000.00|

|4|Chaitali|25|Mumbai|6500.00|

|5|Hardik|27|Bhopal|8500.00|

|6|Komal|22| MP       |4500.00|

|7|Muffy|24|Indore|10000.00|

+----+----------+-----+-----------+----------+
```

Table 2 − ORDERS Table is as follows.

```
+-----+---------------------+-------------+--------+

|OID  | DATE                | CUSTOMER_ID | AMOUNT |

+-----+---------------------+-------------+--------+
|102|2009-10-0800:00:00|3|3000|

|100|2009-10-0800:00:00|3|1500|

|101|2009-11-2000:00:00|2|1560|

|103|2008-05-2000:00:00|4|2060|

+-----+---------------------+-------------+--------+
```

Now, let us join these two tables in our SELECT statement as follows −

```
SQL> SELECT  ID, NAME, AMOUNT, DATE

   FROM CUSTOMERS

   LEFT JOIN ORDERS

   ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID

UNION

   SELECT  ID, NAME, AMOUNT, DATE

   FROM CUSTOMERS

   RIGHT JOIN ORDERS

   ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

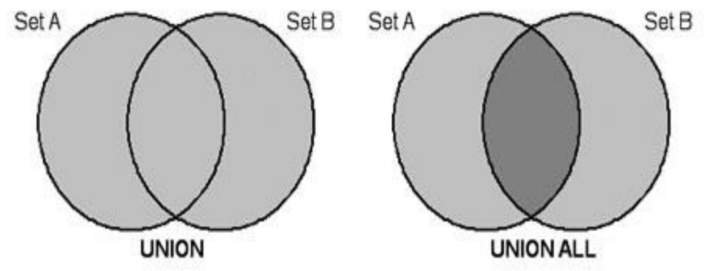
This would produce the following result −

```
+------+----------+--------+---------------------+
| ID   | NAME     | AMOUNT | DATE                |
+------+----------+--------+---------------------+
|    1 | Ramesh   |   NULL | NULL                |
|    2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|    3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|    3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
|    4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
|    5 | Hardik   |   NULL | NULL                |
|    6 | Komal    |   NULL | NULL                |
|    7 | Muffy    |   NULL | NULL                |
+------+----------+--------+---------------------+
```

The UNION ALL Clause

The UNION ALL operator is used to combine the results of two SELECT statements including duplicate rows.

The same rules that apply to the UNION clause will apply to the UNION ALL operator.



Syntax

The basic syntax of the UNION ALL is as follows.

```
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]

UNION ALL

SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
```

Here, the given condition could be any given expression based on your requirement.

Example

Consider the following two tables,

Table 1 − CUSTOMERS Table is as follows.

```
+----+----------+-----+-----------+----------+

| ID | NAME     | AGE | ADDRESS   | SALARY   |

+----+----------+-----+-----------+----------+

|1|Ramesh|32|Ahmedabad|2000.00|

|2|Khilan|25|Delhi|1500.00|

|3| kaushik  |23|Kota|2000.00|

|4|Chaitali|25|Mumbai|6500.00|

|5|Hardik|27|Bhopal|8500.00|
|6|Komal|22| MP       |4500.00|

|7|Muffy|24|Indore|10000.00|

+----+----------+-----+-----------+----------+
```

Table 2 − ORDERS table is as follows.

```
+-----+---------------------+-------------+--------+

|OID  | DATE                | CUSTOMER_ID | AMOUNT |

+-----+---------------------+-------------+--------+

|102|2009-10-0800:00:00|3|3000|

|100|2009-10-0800:00:00|3|1500|

|101|2009-11-2000:00:00|2|1560|

|103|2008-05-2000:00:00|4|2060|

+-----+---------------------+-------------+--------+
```

Now, let us join these two tables in our SELECT statement as follows −

```
SQL> SELECT  ID, NAME, AMOUNT, DATE

  FROM CUSTOMERS

  LEFT JOIN ORDERS

  ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID

UNION ALL

  SELECT  ID, NAME, AMOUNT, DATE

  FROM CUSTOMERS

  RIGHT JOIN ORDERS

  ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

This would produce the following result −

```
+------+----------+--------+---------------------+
| ID   | NAME     | AMOUNT | DATE                |
+------+----------+--------+---------------------+
|    1 | Ramesh   |   NULL | NULL                |
|    2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|    3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|    3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
|    4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
|    5 | Hardik   |   NULL | NULL                |
|    6 | Komal    |   NULL | NULL                |
|    7 | Muffy    |   NULL | NULL                |
|    3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|    3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
|    2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|    4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
+------+----------+--------+---------------------+
```

Note: UNION only appends distinct values. Whereas UNION ALL produces duplicate rows.

Lab Activity:
Students kindly perform the above discussed topics in your lab sessions.

Indexes

- Indexes are special lookup tables that the database search engine can use to speed up data retrieval. Simply put, an index is a pointer to data in a table. An index in a database is very similar to an index in the back of a book.
- For example, if you want to reference all pages in a book that discusses a certain topic, you first refer to the index, which lists all the topics alphabetically and are then referred to one or more specific page numbers.
- An index helps to speed up SELECT queries and WHERE clauses, but it slows down data input, with the UPDATE and the INSERT statements. Indexes can be created or dropped with no effect on the data.
- Creating an index involves the CREATE INDEX statement, which allows you to name the index, to specify the table and which column or columns to index, and to indicate whether the index is in an ascending or descending order.
- Indexes can also be unique, like the UNIQUE constraint, in that the index prevents duplicate entries in the column or combination of columns on which there is an index.

The CREATE INDEX Command

The basic syntax of a CREATE INDEX is as follows.

```
CREATE INDEX index_name ON table_name;
```

1. Single-Column Indexes

A single-column index is created based on only one table column. The basic syntax is as follows.

```
CREATE INDEX index_name
ON table_name (column_name);
```

2. Unique Indexes

Unique indexes are used not only for performance, but also for data integrity. A unique index does not allow any duplicate values to be inserted into the table. The basic syntax is as follows.

```
CREATE UNIQUE INDEX index_name
on table_name (column_name);
```

3. Composite Indexes

A composite index is an index on two or more columns of a table. Its basic syntax is as follows.

```
CREATE INDEX index_name
on table_name (column1, column2);
```

Whether to create a single-column index or a composite index, take into consideration the column(s) that you may use very frequently in a query's WHERE clause as filter conditions.

Should there be only one column used, a single-column index should be the choice. Should there be two or more columns that are frequently used in the WHERE clause as filters, the composite index would be the best choice.

4. Implicit Indexes

Implicit indexes are indexes that are automatically created by the database server when an object is created. Indexes are automatically created for primary key constraints and unique constraints.

The DROP INDEX Command

An index can be dropped using SQL DROP command. Care should be taken when dropping an index because the performance may either slow down or improve.

The basic syntax is as follows −

```
DROP INDEX index_name;
```

You can check the INDEX Constraint chapter to see some actual examples on Indexes.

When should indexes be avoided?

Although indexes are intended to enhance a database's performance, there are times when they should be avoided.

The following guidelines indicate when the use of an index should be reconsidered.

- Indexes should not be used on small tables.
- Tables that have frequent, large batch updates or insert operations.
- Indexes should not be used on columns that contain a high number of NULL values.
- Columns that are frequently manipulated should not be indexed.

SQL TRIGGERS

Introduction to SQL Trigger

- A SQL trigger is executed or fired whenever an event associated with a table occurs e.g., insert, update or delete.
- A SQL trigger is a **special type of stored procedure**. It is special because it is not called directly like a stored procedure.
- **The main difference between a trigger and a stored procedure** is that a trigger is called automatically when a data modification event is made against a table whereas a stored procedure must be called explicitly.
- You can use these triggers on Views, or Tables to perform any of the above specified operations.
- Remember, you can associate a trigger to a **singletable** only.

There are mainly two types of triggers:

1. AFTER TRIGGERS

The after / for triggers will run on trigger after an INSERT, DELETE, or an UPDATEon a table. It means, Before the trigger start running, all the operations should be executed, and the statement has to succeed the constrain check as well. After triggers are not supported on Views so, use them on tables only. It can be further divided into

1. **AFTER INSERT:** This trigger will fire after the completion of Insert operation on table.
2. **AFTER UPDATE:** This trigger will fire after the Update operation is completed on table.
3. **AFTER DELETE:** After delete trigger will fire after the completion of Delete operation on Employee table.

2. INSTEAD OF TRIGGERS

The Instead of Triggers are fired before the execution of an INSERT, DELETE, or an UPDATE on a table start. It means, Before the trigger start running, it does not need any condition check, or constrain check. So, this trigger will execute even the constrain check fails. It can be further divided into

1. INSTEAD OF INSERT: This trigger will fire before the Insert operation on Employee table starts.

2. INSTEAD OF UPDATE: This update trigger will fire before the updating the records in Employee table

3. INSTEAD OF DELETE: This delete trigger will fire before deleting records from Employee table starts.

Syntax of Triggers in SQL Server

1. The basic syntax behind the **After Triggers in SQL Server** is as shown below:

```
-- Create Triggers in SQL Server
CREATE [OR ALTER] TRIGGER [Schema_Name].Trigger_Name
ON Table
AFTER INSERT | UPDATE | DELETE
AS
  BEGIN
    -- Trigger Statements
    -- Insert, Update, Or Delete Statements
  END
```

- **Schema_name:** Please specify the schema name. For example, dbo, or Humanresource etc.
- **Trigger_Name:** You can specify any name you wish to give other than the system reserved keywords. Please try to use meaningful names so that you can identify them easily.

2. The basic syntax behind the **Instead Of Triggers in SQL Server** is as shown below:

```
-- Create Triggers in SQL Server
CREATE [OR ALTER] TRIGGER [Schema_Name].Trigger_Name
ON Table | View
INSTEAD OF INSERT | UPDATE | DELETE
AS
  BEGIN
    -- Trigger Statements
    -- Insert, Update, Or Delete Statements
  END
```

Exercise

Trainer will initiate a discussion ofcommonly interview questions given below:

1: What are advantages of DBMS over traditional file based systems?

2: What are primary and foreign keys?

3: What is the difference between primary key and unique constraints?

4: What is database normalization and their types?

5: What is SQL?

6: What are the differences between DDL, DML and DCL in SQL?

7: What is the difference between having and where clause?

8: What is Join?

9: What is a view in SQL? How to create one?

10: What is a Trigger and stored procedure and their difference?

11: What is a transaction? What are ACID properties?

12: What are indexes? And what are the types of indexes?

13: What is embedded and dynamic SQL?

Lab Activity

Students kindly consider the two tables given below and perform the operations as per the questions given in it.

Table: **employee**

empid	**empname**	**managerid**	**deptid**	**salary**	**DOB**
1	Emp 1	0	1	6000	1982-08-06
2	Emp 2	0	5	6000	1982-07-11
3	Emp 3	1	1	2000	1983-11-21
13	Emp 13	5	5	2000	1984-03-09
3	Emp 3	1	1	2000	1983-11-21
7	Emp 7	5	5	NULL	NULL

Table: **department**

deptid	deptname
1	IT
2	Admin

1.Can you get employee details whose department id is not valid or department id not present in department table?

There are multiple ways to do this.

Using Left JOIN

```
select e.empid,e.empname, e.deptid from employee e
left outer join department d
on e.deptid = d.deptid
where d.deptid is null
```

Using NOT IN

```
select e.empid,e.empname, e.deptid from employee e
where e.deptid not in (select deptid from department)
```

Using NOT Exists

```
select e.empid,e.empname, e.deptid from employee e
where not exists (select deptid from department
where e.deptid=department.deptid)
```

2. Can you get the list of employees with same salary?

With where clause

```
Select distinct e.empid,e.empname,e.salary
from employee e, employee e1
where e.salary =e1.salary
and e.empid != e1.empid
```

3. How can you find duplicate records in Employee table?

```
select empid,empname, salary, count(*) as cnt
from employee
group by empid,empname, salary
having count(*)>1
```

4. How can you DELETE DUPLICATE RECORDS?

There are multiple options to perform this operation.

Using row count to restrict delete only 1 record

```
set rowcount 1
delete from employee where empid in (
select empid
from employee
group by empid,empname, salary
having count(*)>1
)
set rowcount 0
```

5. Find the second highest salary.

```
select max(salary) from employee
where salary not in (select max(salary) from employee)
```

6. Now, can you find 3rd, 5th or 6th i.e. N'th highest Salary?

Query for 3rd highest salary

```
select * from employee e
where 2 = (select count(distinct e1.salary)
from employee e1
where e1.salary>e.salary)
```

Here, 2= 3-1 i.e. N-1 ; can be applied for any number.

7. Can you write a query to find employees with age greater than 30?

```
select * from employee
where datediff(year,dob, getdate()) >30
```

8. Write an SQL Query to print the name of the distinct employees whose DOB is between 01/01/1960 to 31/12/1987

```
select distinct empname from employee
where dob between '01/01/1960' and '12/31/1987'
```

9. Please write a query to get the maximum salary from each department.

```
select deptid, max(salary) as salary from employee group by deptid
```

10. Can you show one row twice in results from a table? (Hint: Cross join can be used)

We can use union all or cross join to obtain this.

```
select deptname from department d where d.deptname='it'
union all
select deptname from department d1 where d1.deptname='it'
```

-- also cross join alias same table

```
select d.deptname from department d, department d1
where d.deptname='it'
```

11. Can you write a query to get employee names starting with a vowel?

Using like operator and expression,

```
Select empid, empname from employee where empname like '[aeiou]%'
```

12. Write a query to get employees who’s ID is an odd number.

```
select * from employee
where empid %2 !=0
```

13. Create view of the Employee table having empid, empname, salary columns only.

```
Create view v_employee
As select empid, empname, salary
From employee;
```

Then:

```
Select * from v_employee;
```

Array

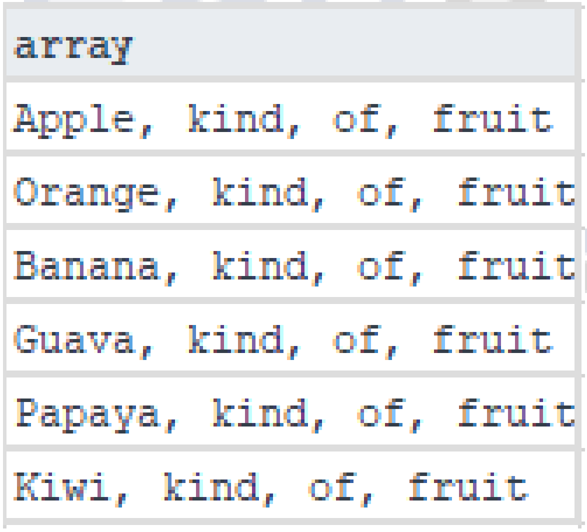
Introduction to Array

- The ARRAY command allows you to create an array from a group of values.
- The values can be field values, calculated values, literals, or a combination thereof.
- The only requirement is that all the values are of the same data type.
- The following is an example of creating an array using the fruit_array table and literals:

Example:

SELECT ARRAY(fruit_id, "kind", "of", “fruit")   FROM fruit_array

A partial set of the results would be:



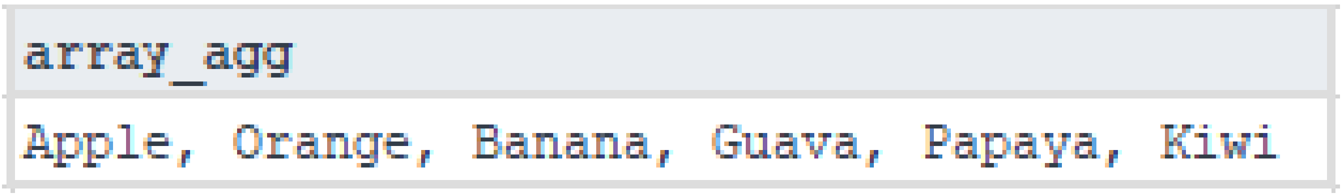
array
Apple, kind, of, fruit
Orange, kind, of, fruit
Banana, kind, of, fruit
Guava, kind, of, fruit
Papaya, kind, of, fruit
Kiwi, kind, of, fruit

ARRAY_AGG

- The sole aggregation for arrays takes all the values in a column and aggregates them into one field.
- For example

  SELECT ARRAY_AGG(fruit_id)   FROM fruit_array

  A partial set of the results would be:

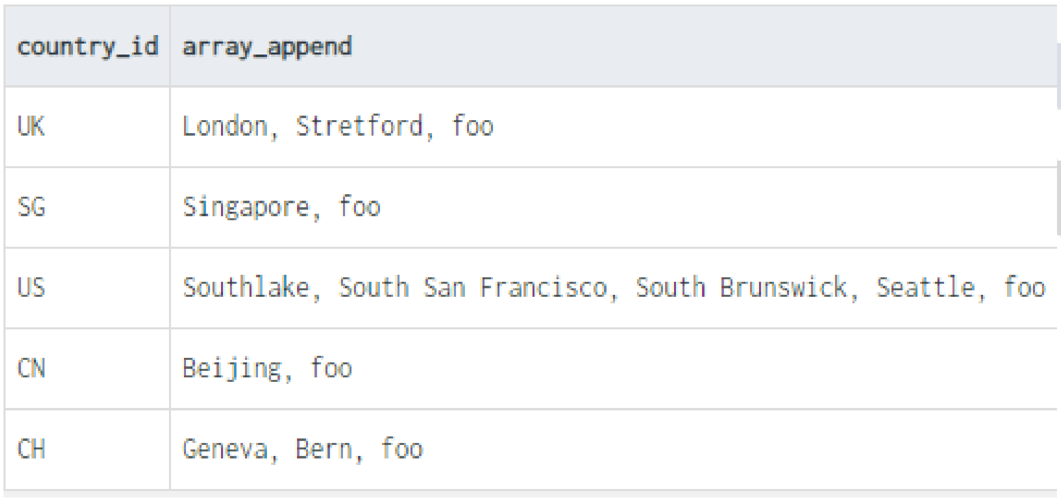


array_agg
Apple, Orange, Banana, Guava, Papaya, Kiwi

ARRAY_APPEND

- The ARRAY_APPEND function is used to append values to an array.
- As with all array functions the values can be field values, calculated values, literals, or a combination thereof.
- The only requirement is that all the values are of the same data type.
- The following is an example of a query using ARRAY_APPEND:

```
SELECT country_id,  ARRAY_APPEND(cities, "foo")   FROM city_array
```



country_id	array_append
UK	London, Stretford, foo
SG	Singapore, foo
US	Southlake, South San Francisco, South Brunswick, Seattle, foo
CN	Beijing, foo
CH	Geneva, Bern, foo

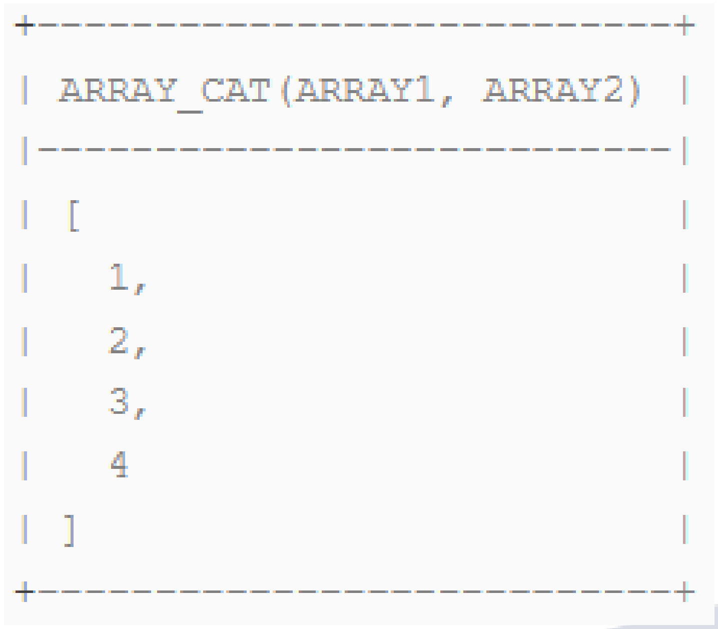
ARRAY_CAT

- The ARRAY_CAT function is used to concatenate two arrays.
- The following is an example of a query using ARRAY_APPEND:

```
array1 = [1,2,3]
```

array2 = [4]

select array_cat(array1, array2) from array_demo;

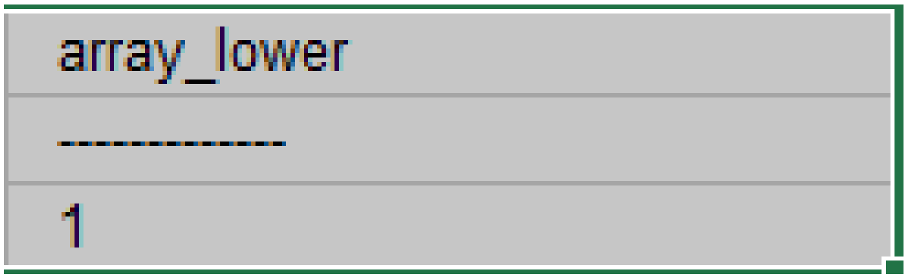


```
+---------------------------+
| ARRAY_CAT(ARRAY1, ARRAY2) |
|---------------------------|
| [                         |
|   1,                      |
|   2,                      |
|   3,                      |
|   4                       |
| ]                         |
+---------------------------+
```

ARRAY_LOWER

- This function is used to return the lower bound of the requested array dimension.
- The following is an example of a query using ARRAY_LOWER:

SELECT array_lower(ARRAY[0, 1, 2], 1);



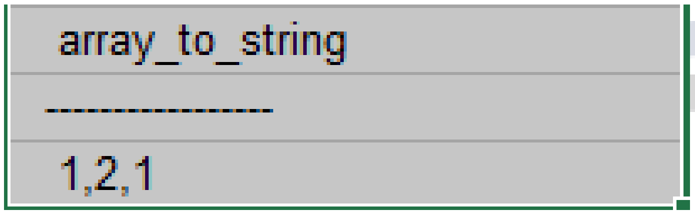
```
array_lower
--------------
1
```

This means that the starting subscript of [0, 1, 2] is 1.

ARRAY_to_string

- This function is used to concatenate array elements using supplied delimiter and optional null string.
- The following is an example of a query using ARRAY_TO_STRING:

```
SELECT array_to_string(ARRAY[1, NULL, 2, 1], ',');
```



```
array_to_string
-----------------
1,2,1
```

This example shows how to use the array_to_string() function to convert {1, NULL, 2, 1} to a comma-separated string.

Note that the NULL element in the array is ignored. If you want to output the NULL element as a specific value, for example 0, use the following statement: SELECT array_to_string(ARRAY[1, NULL, 2, 1], ',', '0');

Comment

- Comments can make your application easier for you to read and maintain.
- For example, you can include a comment in a statement that describes the purpose of the statement within your application.

- With the exception of hints, comments within SQL statements do not affect the statement execution.

Single Line Comments

- ✓ Single line comments start with --.
- ✓ Any text between -- and the end of the line will be ignored (will not be executed).

```
--Select all:
SELECT * FROM Customers
```

Multi Line Comments

- ✓ Multi-line comments start with /* and end with */.
- ✓ Any text between /* and */ will be ignored.

```
/*Select all the columns
of all the records
in the Customers table:*/
SELECT * FROM Customers;
```