Unit – IV Introduction to SQL

SQL

- SQL stands for Structured Query Language. It is used for storing and managing data in relational database management system (RDMS).
- It is a standard language for Relational Database System. It enables a user to create, read, update and delete relational databases and tables.
- All the RDBMS like MySQL, Informix, Oracle, MS Access and SQL Server use SQL as their standard database language.
- SQL allows users to query the database in a number of ways, using English-like statements.

Rules:

SQL follows the following rules:

- Structure query language is not case sensitive. Generally, keywords of SQL are written in uppercase.
- Statements of SQL are dependent on text lines. We can use a single SQL statement on one or multiple text line.
- Using the SQL statements, you can perform most of the actions in a database.
- SQL depends on tuple relational calculus and relational algebra.

SQL process:

- When an SQL command is executing for any RDBMS, then the system figure out the best way to carry out the request and the SQL engine determines that how to interpret the task.
- In the process, various components are included. These components can be optimization Engine, Query engine, Query dispatcher, classic, etc.
- All the non-SQL queries are handled by the classic query engine, but SQL query engine won't handle logical files.

Basic Data Types

- Number
- Char
- Varchar and Varchar2
- Date and Time

NUMBER Datatype

The `NUMBER` datatype stores fixed and floating-point numbers. Numbers of virtually any magnitude can be stored and are guaranteed portable among different systems operating Oracle Database, up to 38 digits of precision.

The following numbers can be stored in a `NUMBER` column:

- Positive numbers
- Negative numbers
- Zero

For numeric columns, you can specify the column as:

```
column_name NUMBER
```

Optionally, you can also specify a precision (total number of digits) and scale (number of digits to the right of the decimal point)

```
column_name NUMBER (precision, scale)
```

If a precision is not specified, the column stores values as given. If no scale is specified, the scale is zero.

Oracle guarantees portability of numbers with a precision equal to or less than 38 digits. You can specify a scale and no precision:

```
column_name NUMBER (*, scale)


Examples of how data would be stored

using different scale factors. How Scale Factors Affect Numeric Data Storage
```

Input Data	**Specified As**	**Stored As**
7,456,123.89	NUMBER	7456123.89
7,456,123.89	NUMBER(*,1)	7456123.9
7,456,123.89	NUMBER(9)	7456124
7,456,123.89	NUMBER(9,2)	7456123.89
7,456,123.89	NUMBER(9,1)	7456123.9
7,456,123.89	NUMBER(6)	(not accepted, exceeds precision)
7,456,123.89	NUMBER(7,-2)	7456100

DDL

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 the database.

- DDL stands for **Data Definition Language.**
- It is a language used for defining and modifying the data and its structure.
- It is used to build and modify the structure of your tables and other objects in the database.

SQLs data definition language (DDL) defines information about each relation in a database which is listed below in detail:

- SQLs DDL specifies the **schema** of each relation i.e. the **logical design** of each relation which states the **name** of that relation, **attributes** it carry and also specifies the **domain** of those attributes.
- SQL DDL specifies the **integrity constraint** which makes sure that any changes made to the database don't affect the consistency of data.
- SQL DDL also maintains **the set of indices** for each relation which let you retrieve the records from the database quickly.
- SQL DDL maintains the information about the **security** of data in the database and it also keeps the information regarding the **authorization** for each relation in the database.
- SQL DDL also describes the **storage structure** of each relation on the hard disk.

Examples of DDL commands:

- **CREATE** – is used to create the database or its objects (like table, index, function, views, store procedure and triggers).
- **DROP** – is used to delete objects from the database.
- **ALTER**-is used to alter the structure of the database.
- **TRUNCATE**–is used to remove all records from a table, including all spaces allocated for the records are removed.
- **COMMENT** –is used to add comments to the data dictionary.
- **RENAME** –is used to rename an object existing in the database.

DDL Commands

1. CREATE COMMAND

- **CREATE command** is used for creating objects in the database.
- It creates a new table.

Syntax:

```
CREATE TABLE <table_name>
(    column_name1 datatype,
     column_name2 datatype,
     .
     .
     .
     column_name_n datatype
);
```

Example : CREATE command

```
CREATE TABLE employee
(
     empid INT,
     ename CHAR,
     age INT,
     city CHAR(25),
     phone_no VARCHAR(20)
);
```

2. DROP COMMAND

- **DROP command** allows to remove entire database objects from the database.
- It removes entire data structure from the database.
- It deletes a table, index or view.

Syntax:

```
DROP TABLE <table_name>;
OR
DROP DATABASE <database_name>;
```

Example : DROP Command

```
DROP TABLE employee;
OR
DROP DATABASE employees;
```

- If you want to remove individual records, then use DELETE command of the DML statement.

3. ALTER COMMAND

- An **ALTER command** allows to alter or modify the structure of the database.
- It modifies an existing database object.
- Using this command, you can add additional column, drop existing column and even change the data type of columns.

Syntax:

```
ALTER TABLE <table_name>
ADD <column_name datatype>;

OR

ALTER TABLE <table_name>
CHANGE <old_column_name> <new_column_name>;

OR

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

Example : ALTER Command

```
ALTER TABLE employee
ADD (address varchar2(50));

OR

ALTER TABLE employee
CHANGE (phone_no) (contact_no);
```

OR

ALTER TABLE employee
DROP COLUMN age;

To view the changed structure of table, use 'DESCRIBE' command.
For example:
DESCRIBE TABLE employee;

4. RENAME COMMAND

- **RENAME command** is used to rename an object.
- It renames a database table.

Syntax:
RENAME TABLE <old_name> TO <new_name>;

Example:
RENAME TABLE emp TO employee;

5. TRUNCATE COMMAND

- **TRUNCATE command** is used to delete all the rows from the table permanently.
- It removes all the records from a table, including all spaces allocated for the records.
- This command is same as DELETE command, but TRUNCATE command does not generate any rollback data.

Syntax:
TRUNCATE TABLE <table_name>;

Example:
TRUNCATE TABLE employee;

Queries

The SQL **SELECT** statement is used to fetch the data from a database table which returns this data in the form of a result table. These result tables are called result-sets.

Syntax

The basic syntax of the SELECT statement is as follows −

```
SELECT column1, column2, columnN FROM table_name;
```

Here, column1, column2... are the fields of a table whose values you want to fetch. If you want to fetch all the fields available in the field, then you can use the following syntax.

```
SELECT * FROM table_name;
```

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 |
+----+----------+-----+-----------+----------+
```

The following code is an example, which would fetch the ID, Name and Salary fields of the customers available in CUSTOMERS table.

```
SQL> SELECT ID, NAME, SALARY FROM CUSTOMERS;
```

This would produce the following result −

```
+----+----------+----------+
| ID | NAME     | SALARY   |
+----+----------+----------+
|  1 | Ramesh   |  2000.00 |
|  2 | Khilan   |  1500.00 |
|  3 | kaushik  |  2000.00 |
|  4 | Chaitali |  6500.00 |
|  5 | Hardik   |  8500.00 |
|  6 | Komal    |  4500.00 |
|  7 | Muffy    | 10000.00 |
+----+----------+----------+
```

If you want to fetch all the fields of the CUSTOMERS table, then you should use the following query.

```
SQL> SELECT * FROM CUSTOMERS;
```

This would produce the result as shown below.

```
+----+----------+-----+-----------+----------+
| 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 |
+----+----------+-----+-----------+----------+
```

INSERT & UPDATE STATEMENT

The SQL **INSERT INTO** Statement is used to add new rows of data to a table in the database.

Syntax

There are two basic syntaxes of the INSERT INTO statement which are shown below.

```
INSERT INTO TABLE_NAME (column1, column2, column3,...columnN)
VALUES (value1, value2, value3,...valueN);
```

Here, column1, column2, column3,...columnN are the names of the columns in the table into which you want to insert the data.

You may not need to specify the column(s) name in the SQL query if you are adding values for all the columns of the table. But make sure the order of the values is in the same order as the columns in the table.

The SQL INSERT INTO syntax will be as follows −

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

Example

The following statements would create six records in the CUSTOMERS table.

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (1, 'Ramesh', 32, 'Ahmedabad', 2000.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (2, 'Khilan', 25, 'Delhi', 1500.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (3, 'kaushik', 23, 'Kota', 2000.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (4, 'Chaitali', 25, 'Mumbai', 6500.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (5, 'Hardik', 27, 'Bhopal', 8500.00 );
```

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (6, 'Komal', 22, 'MP', 4500.00 );
```

You can create a record in the CUSTOMERS table by using the second syntax as shown below.

```
INSERT INTO CUSTOMERS
VALUES (7, 'Muffy', 24, 'Indore', 10000.00 );
```

All the above statements would produce the following records in the CUSTOMERS table as shown below.

```
+----+----------+-----+-----------+----------+
| 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 |
+----+----------+-----+-----------+----------+
```

The UPDATE statement is used to modify the existing records in a table.

UPDATE Syntax

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

UPDATE Table

The following SQL statement updates the first customer (CustomerID = 1) with a new contact person *and* a new city.

```
UPDATE Customers
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
WHERE CustomerID = 1;
```

UPDATE Multiple Records

It is the WHERE clause that determines how many records will be updated.

The following SQL statement will update the contactname to "Juan" for all records where country is "Mexico":

```
UPDATE Customers
SET ContactName='Juan'
WHERE Country='Mexico';
```

Delete statement

The SQL DELETE Query is used to delete the existing records from a table.

You can use the WHERE clause with a DELETE query to delete the selected rows, otherwise all the records would be deleted.

Syntax

The basic syntax of the DELETE query with the WHERE clause is as follows −

```
DELETE FROM table_name
WHERE [condition];
```

You can combine N number of conditions using AND or OR operators.

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 |
+----+----------+-----+-----------+----------+
```

The following code has a query, which will DELETE a customer, whose ID is 6.

```
SQL> DELETE FROM CUSTOMERS
WHERE ID = 6;
```

Now, the CUSTOMERS table would have 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 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```
The following SQL statement deletes the customer "Alfreds Futterkiste" from the "Customers" table:

Example

DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';

If you want to DELETE all the records from the CUSTOMERS table, you do not need to use the WHERE clause and the DELETE query would be as follows

```
SQL> DELETE FROM CUSTOMERS;
```
Now, the CUSTOMERS table would not have any record.

View in SQL

A view is nothing more than a SQL statement that is stored in the database with an associated name. A view is actually a composition of a table in the form of a predefined SQL query.

A view can contain all rows of a table or select rows from a table.

A view can be created from one or many tables which depends on the written SQL query to create a view.

Views, which are a type of virtual tables allow users to do the following −

- Structure data in a way that users or classes of users find natural or intuitive.
- Restrict access to the data in such a way that a user can see and (sometimes) modify exactly what they need and no more.
- Summarize data from various tables which can be used to generate reports.

Creating Views

The basic **CREATE VIEW** syntax is as follows −

CREATE VIEW view_name AS

SELECT column1, column2.....

```
FROM table_name

WHERE [condition];
```

You can include multiple tables in your SELECT statement in a similar way as you use them in a normal SQL SELECT query.

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 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 |

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

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.

The following code block has an example to update the age of Ramesh.

```
SQL > UPDATE CUSTOMERS_VIEW

   SET AGE = 35

   WHERE name = 'Ramesh';
```

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.

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 |

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

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;

Constraint

Constraints are the rules enforced on the data columns of 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.

Constraints could be either on a column level or a table level. The column level constraints are applied only to one column, whereas the table level constraints are applied to the whole table.

Following are some of the most commonly used constraints available in SQL. These constraints have already been discussed in **SQL - RDBMS Concepts** chapter, but it’s worth to revise them at this point.

- **NOT NULL Constraint** − Ensures that a column cannot have NULL value.
- **DEFAULT Constraint** − Provides a default value for a column when none is specified.
- **UNIQUE Constraint** − Ensures that all values in a column are different.
- **PRIMARY Key** − Uniquely identifies each row/record in a database table.

- FOREIGN Key – Uniquely identifies a row/record in any of the given database table.
- CHECK Constraint – The CHECK constraint ensures that all the values in a column satisfies certain conditions.
- INDEX – Used to create and retrieve data from the database very quickly.

Constraints can be specified when a table is created with the CREATE TABLE statement or you can use the ALTER TABLE statement to create constraints even after the table is created.

Dropping Constraints

Any constraint that you have defined can be dropped using the ALTER TABLE command with the DROP CONSTRAINT option.

For example, to drop the primary key constraint in the EMPLOYEES table, you can use the following command.

```
ALTER TABLE EMPLOYEES DROP CONSTRAINT EMPLOYEES_PK;
```

Some implementations may provide shortcuts for dropping certain constraints. For example, to drop the primary key constraint for a table in Oracle, you can use the following command.

```
ALTER TABLE EMPLOYEES DROP PRIMARY KEY;
```

Some implementations allow you to disable constraints. Instead of permanently dropping a constraint from the database, you may want to temporarily disable the constraint and then enable it later.

Integrity Constraints

Integrity constraints are used to ensure accuracy and consistency of the data in a relational database. Data integrity is handled in a relational database through the concept of referential integrity.

There are many types of integrity constraints that play a role in **Referential Integrity (RI)**. These constraints include Primary Key, Foreign Key, Unique Constraints and other constraints which are mentioned above.

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

Additional Features

- SQL is easy to learn.
- SQL is used to access data from relational database management systems.
- SQL can execute queries against the database.
- SQL is used to describe the data.
- SQL is used to define the data in the database and manipulate it when needed.
- SQL is used to create and drop the database and table.
- SQL is used to create a view, stored procedure, function in a database.
- SQL allows users to set permissions on tables, procedures, and views.

Advantages of SQL

There are the following advantages of SQL:

High speed

Using the SQL queries, the user can quickly and efficiently retrieve a large amount of records from a database.

No coding needed

In the standard SQL, it is very easy to manage the database system. It doesn't require a substantial amount of code to manage the database system.

Well defined standards

Long established are used by the SQL databases that are being used by ISO and ANSI.

Portability

SQL can be used in laptop, PCs, server and even some mobile phones.

Interactive language

SQL is a domain language used to communicate with the database. It is also used to receive answers to the complex questions in seconds.

Multiple data view

Using the SQL language, the users can make different views of the database structure.