Unit - 2:

Basic SQL: Introduction to SQL, Basic SQL Queries: DML, DDL, DCL, TCL

SQL(Structured Query Language)

- Structured Query Language (SQL) is the most widely used commercial relational database language.

- Different aspects of SQL language.
  - The Data Manipulation Language (DML): This subset of SQL allows users to pose queries and to insert, delete, and modify rows.
  - The Data Definition Language (DDL): This subset of SQL supports the creation, deletion, and modification of definitions for tables and views. Integrity constraints can be defined on tables, either when the table is created or later.
  - Triggers and Advanced Integrity Constraints: Triggers are actions executed by the DBMS whenever changes to the database meet conditions specified in the trigger. SQL allows the use of queries to specify complex integrity constraint specifications.
  - Embedded and Dynamic SQL: Embedded SQL features allow SQL code embedded to any programming languages, such a language is called as a host language such as C or COBOL. Dynamic SQL features allow a query to be constructed (and executed) at run-time.
  - Client-Server Execution and Remote Database Access: These commands control how a client application program can connect to an SQL database server, or access data from a database over a network.
  - Transaction Management: Various commands allow a user to explicitly control aspects of how a transaction is to be executed.
  - Security: SQL provides mechanisms to control users access to data objects such as tables and views.
  - Advanced features: The SQL:1999 standard includes object-oriented feature, recursive queries, decision support queries.

- DBMS is software that defines different operations to be carried out in the database. It varies from creating a database, tables, index, constraints to manipulating the data in the database like inserting, deleting, updating, retrieving, sorting etc. In order to perform all these operations, DBMS defines the forms of database languages.

- Below are the database languages.

DDL	Data Definition Language
DML	Data Manipulation Language
DCL	Data Control Language
TCL	Transaction Control Language

DDL (Data Definition Language)

- DDLs are used to define the metadata of the database. i.e.; using this, we create schema, tables, constraints, indexes in the database.
- DDLs are also used to modify Schema, tables index etc. Basically, using DDL statements, we create skeleton of the database.
- It helps to store the metadata information like number of schemas and tables, their names, columns in each table, indexes, constraints etc in the database. DDL commands are as follows
  - To create the database instance – CREATE
  - To alter the structure of database – ALTER
  - To drop database instances – DROP
  - To delete tables in a database instance – TRUNCATE
  - To rename database instances – RENAME

- CREATE : Create is used to create schema, tables, index, and constraints in the database. The basic syntax to create table is as follows.

```
CREATE TABLE tablename
(
Column1 DATATYPE,
Column2 DATATYPE,
 ...
ColumnN DATATYPE
);
```

EXAMPLE: CREATE TABLE STUDENT

```
(
STUDENT_ID CHAR (10),
STUDENT_NAME CHAR (10)
);
```

- Constraints are created along with table creation. Constraints are defined on the columns of the table. They define the characteristic of the column. There are different types of Constraints present.

- NOT NULL – This constraint forces the column to have non-null value. We cannot enter/update any NULL value into such columns. It must have valid value all the time. Example: each student in STUDENT table should have class specified. No student can exist without class. Hence class column in the STUDENT table can be made NOT NULL.

  ```
  CREATE TABLE STUDENT (

  STUDENT_ID NUMBER (10) NOT NULL,
  STUDENT_NAME VARCHAR(50) NOT NULL, AGE
  NUMBER

  );
  ```

- UNIQUE – This constraint ensures, the column will have unique value for each row. The column value will not repeat for any other rows in the table.
  Example: Passport number of individual person is unique. Hence passport column in the PERSON table is made UNIQUE. It avoids duplicate entry of passport number to other persons.

  ```
  CREATE TABLE STUDENT (

  STUDENT_ID NUMBER (10) NOT NULL UNIQUE,
  STUDENT_NAME VARCHAR2 (50) NOT NULL, AGE
  NUMBER

  );
  OR

  CREATE TABLE STUDENT (

  STUDENT_ID NUMBER (10) NOT NULL,
  STUDENT_NAME VARCHAR (50) NOT NULL, AGE
  NUMBER

  CONSTRAINT uc_StdID UNIQUE (STUDENT_ID)
  );
  ```

- PRIMARY KEY – This constraint is another type of UNIQUE constraint. This constraint forces the column to have unique value and using which, we can uniquely determine each row.
  Example: As we have seen in STUDENT example, STUDENT_ID is the primary key in STUDENT tables.

  ```
  CREATE TABLE STUDENT (

  STUDENT_ID NUMBER (10) NOT NULL PRIMARY KEY, STUDENT_NAME
  VARCHAR (50) NOT NULL,

  AGE NUMBER

  );
  OR

  CREATE TABLE STUDENT (
  ```

```
STUDENT_ID NUMBER (10) NOT NULL,
STUDENT_NAME VARCHAR(50) NOT NULL, AGE
NUMBER

CONSTRAINT pk_StdID PRIMARY KEY (STUDENT_ID)
);
```

- FOREIGN KEY – This constraint helps to map two or more tables in the database. It enforces parent-child relationship in the DB. Foreign key in the child table is the column which is a primary key in the parent table.

  Example: each employee works for some department. Hence to map employee and department tables, we have to have DEPARMENT_ID of DEPARTMENT table in EMPLOYEE table too. DEPARTMENT_ID is the primary key in DEPARTMENT table (Parent table) and is foreign key in EMPLOYEE table (Child table).

```
CREATE TABLE EMPLOYEE (

EMPLOYEE_ID VARCHAR(10) PRIMARY KEY,
EMP_NAME VARCHAR2 (50),

DOB DATE, DEPT_ID
NUMBER

FOREIGN KEY (DEPT_ID) REFERENCES DEPARTMENT (DEPARTMENT_ID)

);
```

- CHECK – This constraint is used to check for specific conditions on the column. For example, if age has to be entered between 25 and 32, we can use CHECK Constraint. This will not allow to enter the age 32.

```
CREATE TABLE STUDENT (

STUDENT_ID NUMBER (10) NOT NULL,
STUDENT_NAME VARCHAR(50) NOT NULL,
AGE NUMBER CHECK (AGE >= 25 and AGE<= 32)
);
```

- DEFAULT – This constraint specifies the default value to be entered when no value is entered to it. Suppose whenever we enter an entry in the STUDENT table, apart from Student details we also have to store the date when it is being entered. This entry would always be SYSDATE. Instead of entering it each time when we do an entry, if we set the default value of this column as SYSDATE, this column will be always inserted with SYSDATE. If we need to override this value with any other date, then we have to explicitly insert the new date.

```
CREATE TABLE STUDENT (

STUDENT_ID NUMBER (10) NOT NULL,
STUDENT_NAME VARCHAR(50) NOT NULL,
```

AGE NUMBER,

CURRENT_DATE DATE DEFAULT SYSDATE

);

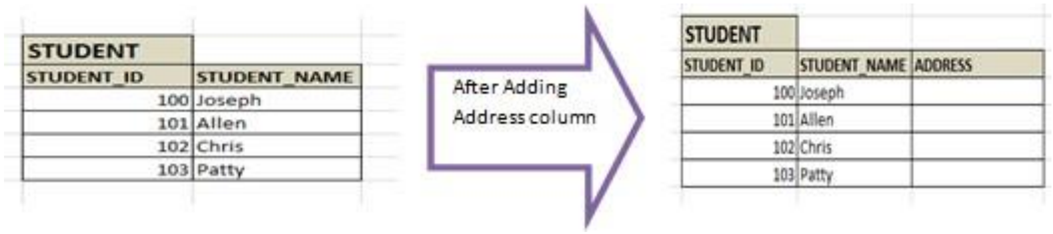
- ALTER

Suppose we have created a STUDENT table with his ID and Name. Later we realize that this table should have his address and Age too. What do we do at this stage? We will add the column to the existing table by the use of ALTER command.

- This command is used to modify the structure of the Schema or table. It can even used to add, modify or delete the columns in the table. The syntax for alter statement is as follows.
  To add a new column: ALTER TABLE table_name ADD column_namedatatype;
  To delete a column: ALTER TABLE table_name DROP COLUMN column_name;
  To modify a column: ALTER TABLE table_name MODIFY column_namedatatype;
  To rename table: ALTER TABLE table_name RENAME TO new_table_name;
  To rename the column: ALTER TABLE table_name RENAME COLUMN
  old_Column_name to new_Column_name;

- Suppose we want to add Address column to the STUDENT table.
  ALTER TABLE STUDENT ADD Address varchar2 (100);

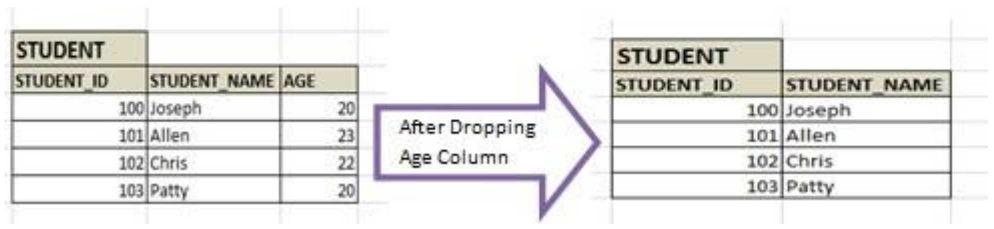
Once we add new columns to the existing table, the column value for the existing data would be NULL. If we need value in them, either we have to set some default value or we need to explicitly update each column with proper value.


After Adding
Address column

STUDENT	
STUDENT_ID	STUDENT_NAME
100	Joseph
101	Allen
102	Chris
103	Patty

STUDENT		
STUDENT_ID	STUDENT_NAME	ADDRESS
100	Joseph	
101	Allen	
102	Chris	
103	Patty	

- Suppose we want to Drop Age column from STUDENT table.
  ALTER TABLE STUDENT DROP COLUMN AGE;

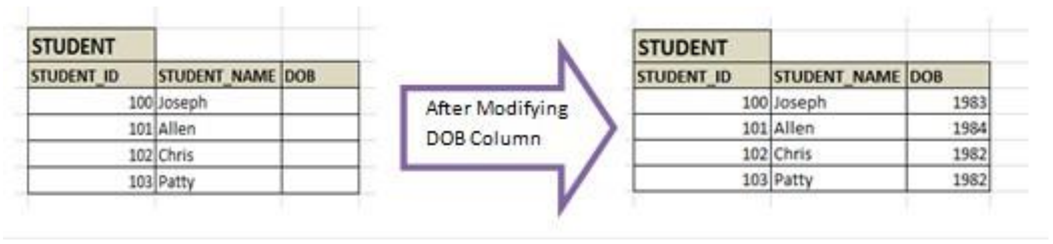

After Dropping
Age Column

STUDENT		
STUDENT_ID	STUDENT_NAME	AGE
100	Joseph	20
101	Allen	23
102	Chris	22
103	Patty	20

STUDENT	
STUDENT_ID	STUDENT_NAME
100	Joseph
101	Allen
102	Chris
103	Patty

Once column is dropped, the entire information in the column is lost.

- Suppose we want to modify the DOB column to have only year in it.
  ALTER TABLE STUDENT MODIFY DOB NUMBER (4);


After Modifying
DOB Column

STUDENT		
STUDENT_ID	STUDENT_NAME	DOB
100	Joseph	
101	Allen	
102	Chris	
103	Patty	

STUDENT		
STUDENT_ID	STUDENT_NAME	DOB
100	Joseph	1983
101	Allen	1984
102	Chris	1982
103	Patty	1982

- Note: In order to change the one datatype to another datatype, that particular column should not have any value.
- If we are changing the length of the column, then we can do this with data in the column.
- If the length of the column is reduced, then the value in the column will be trimmed to adjust with new length.
- Suppose name column is modified from Varchar (20) to Varchar(10) and one of the name was 'Albert Einstein'. After column modification, the name would be automatically trimmed to 'Albert Ein'

- DROP: - DROP statement is used to remove the table or index from the database. It can even be used to remove the database. Once the DROP statement is executed, the object will not available for use.
  DROP TABLE table_name;
  DROP DATABASE database_name;
  DROP TABLE STUDENT;

- Truncate
  Truncate statement is used to remove the content of the table, but keeps the structure of the table. This simply removes all the records from the table. No partial removal of data is possible here. It also removes all the spaces allocated for the data.
  TRUNCATE TABLE table_name;
  TRUNCATE TABLE STUDENT;

DML (Data Manipulation Language)

- When we have to insert records into table or get specific record from the table, or need to change some record, or delete some record or perform any other actions on records in the database, we need to have some media to perform it.
- DML helps to handle user requests. It helps to insert, delete, update, and retrieve the data from the database.

- SELECT

Select command helps to pull the records from the tables or views in the database. It either pulls the entire data from the table/view or pulls specific records based on the condition. It can even retrieve the data from one or more tables/view in the database.

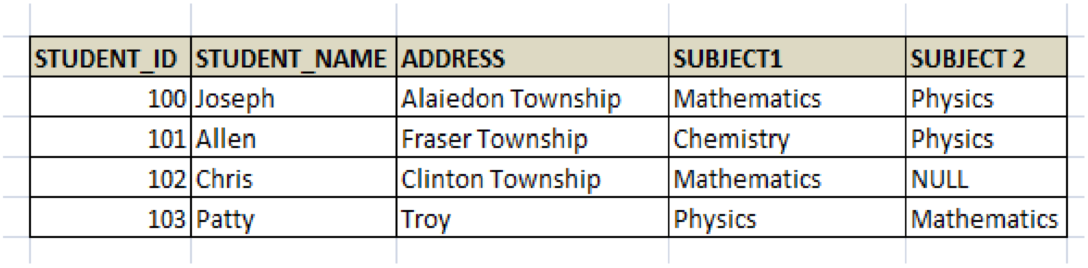
The basic SELECT command is
SELECT * FROM table_name; -- retrieves all the rows and columns from table table_name and displays it in tabular form.

SELECT COLUMN1, COLUMN2, COLUMN3 from table_name; -- retrieves only 3 columns from table table_name

SELECT t1.COLUMN1, t2.COLUMN1
FROM table_name1 t1, table_name2 t2
WHERE t1.COLUMN2 = t2.COLUMN2;

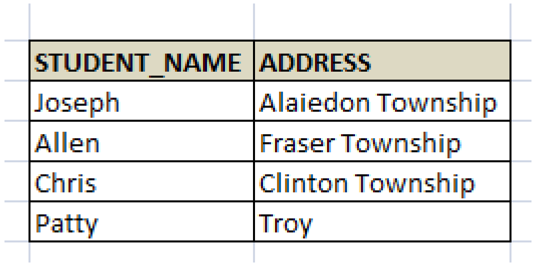
Some examples of SELECT:

- All the columns are retrieved and displayed in below format
  SELECT * FROM STUDENT;



STUDENT_ID	STUDENT_NAME	ADDRESS	SUBJECT1	SUBJECT 2
100	Joseph	Alaiedon Township	Mathematics	Physics
101	Allen	Fraser Township	Chemistry	Physics
102	Chris	Clinton Township	Mathematics	NULL
103	Patty	Troy	Physics	Mathematics

- Retrieves only name and address from STUDENT table
  SELECT STUDENT_NAME, ADDRESS FROM STUDENT;



STUDENT_NAME	ADDRESS
Joseph	Alaiedon Township
Allen	Fraser Township
Chris	Clinton Township
Patty	Troy

- Advanced Select Commands
  General syntax of SELECT is:
  SELECT column_list FROM table-name
  [WHERE Clause]
  [GROUP BY clause]
  [HAVING clause]
  [ORDER BY clause];
  - WHERE Clause - here we can specify the filter conditions to the query. We can add any number of conditions.

    - Example: there are multiple employees working in different department. Using where option we can display employees are working in a department.
      SELECT d.DEPATMENT_NAME, e.EMPLOYEE_NAME
      FROM EMPLOYEE e, DEPARTMENT d
      WHERE e.DEPARTMENT_ID = d.DEPARTMENT_ID;

- GROUP BY - We can combine specific categories of columns together and show the results.
    - Example: there are multiple employees working in different department. Using group by option we can display how many employees are working in each department.
      SELECT d.DEPATMENT_NAME, e.EMPLOYEE_NAME
      FROM EMPLOYEE e, DEPARTMENT d
      WHERE e.DEPARTMENT_ID = d.DEPARTMENT_ID
      GROUP BY e.DEPARTMENT_ID;

- ORDER BY - This clause helps to sort the records that are retrieved. By default, it displays the records in ascending order of primary key. If we need to sort it based on different columns, then we need to specify it in ORDER BY clause. If we need to order by descending order, then DESC keyword or we need to order by ascending order, then ASC keyword has to be added after the column list.
  Example: this display all records in employee name column in descending order.

  ```
  SELECT * FROM EMPLOYEE

  ORDER BY EMPLOYEE_NAME DESC;
  ```

- Combine two or more columns into one column by using || in select statement.
  SELECT EMP_FIRST_NAME || ‘ ‘ || EMP_LAST_NAME as emp_name
  FROM EMPLOYEE WHERE EMP_ID = 1001;
  (Here first name and last name of employee with id 1001 to show it as emp_name)

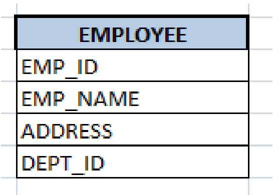
- INSERT : Insert statement is used to insert new records into the table.
  The general syntax for insert is as follows:
  INSERT INTO TABLE_NAME (col1, col2, col3,...colN)
  VALUES (value1, value2, value3,...valueN);

    - It inserts value to each column from 1 to N. When we insert the data in this way, we need to make sure datatypes of each column matches with the value we are inserting. Else, data will not be inserted or will insert wrong values.
    - Also, if there is any foreign key constraint on the table, then we have to make sure foreign key value already exists in the parent table.

```
INSERT INTO EMPLOYEE (EMP_ID, EMP_NAME, ADDRESS, DEPT_ID)
```

VALUES (10001, ‘Joseph’, ‘Troy’, 11101);

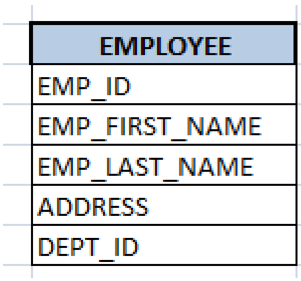
- Here dept_id 11101 is a foreign key and it has be inserted into department table before inserting into the employee table.
- We can insert the values into the table without specifying the columns, provided we enter the value in the order the table structure is. Imagine table structure for Employee is as follows and then we can insert the record without specifying column names as below:



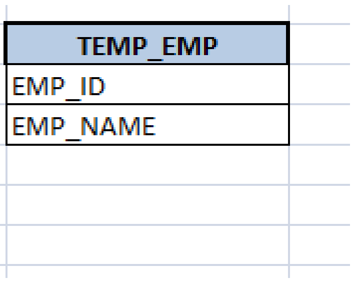
EMPLOYEE
EMP_ID
EMP_NAME
ADDRESS
DEPT_ID

INSERT INTO EMPLOYEE VALUES (10001, ‘Joseph’, ‘Troy’, 11101);

- If we are specifying the column list then we need not insert it in the order of table structure. It inserts the values in the order the column is listed in the INSERT statement.
  INSERT INTO EMPLOYEE (EMP_ID, DEPT_ID, ADDRESS, EMP_NAME) VALUES (10001, 11101,’Troy’, ‘Joseph’);
- We can even copy some of the column values from the existing table. Suppose we have to copy emp_id, emp_name from Employee table to some temporary table. Then we can write as follows: (Assuming here that datatype in both the tables are same and emp_name has enough buffers to store both first name and last name combined).



EMPLOYEE
EMP_ID
EMP_FIRST_NAME
EMP_LAST_NAME
ADDRESS
DEPT_ID



TEMP_EMP
EMP_ID
EMP_NAME

- Update
  - Update statement is used to modify the data value in the table. General syntax for update is as below:

    UPDATE table_name
    SET column_name1 = value1,
    column_name2 = value2,
    ...

    column_nameN = valueN,
    [WHERE condition]

  - Imagine, an employee has changed his address and it needs to be updated in the Employee table.

```
UPDATE EMPLOYEE
SET ADDRESS = 'Clinton Township'
WHERE EMP_ID = 10110;
```

- If we do not specify 'WHERE' condition in the 'UPDATE' statement, then it will update the whole table with new address. Hence we have to specify which record/employee has to be updated with new address.
- Suppose there is increment in the salary of all the employees by 10% and this has to be updated in the Employee table.
  UPDATE EMPLOYEE SET Salary = salary+ (salary*0.1);

- Delete
  - Using Delete statement, we can delete the records in the entire table or specific record by specifying the condition.
    DELETE FROM table_name [WHERE condition];
  - Suppose we have to delete an employee with id 110 from Employee table. Then the delete statement would be
    DELETE FROM EMPLOYEE WHERE EMP_ID = 110;
  - If we do not specify the condition, then it would delete entire record from the Employee table. This statement is different from TRUNCATE in two ways:
  - Using DELETE statement, we can delete few records by specifying the condition. If we do not specify the condition, it deletes entire records in table. Whereas TRUNCATE deletes all the records in the table.
  - DELETE statement simply removes records from the table, whereas TRUNCATE statement frees the space occupied by the data. Hence TRUNCATE is more efficient than DELETE, when we have to empty the table.

- Aggregate Functions
  Some of the Group functions are used to perform some computation or summarization those functions are called as aggregate functions.
  - Count - it counts the total number of records in the table/s after applying 'where' clause. If where clause is not specified, it gives the total number of records in the table. If Group by clause is applied, it filters the records based on where clause (if any), then groups the records based on the columns in group by clause and gives the total count of records in each grouped category.
  - SUM – It totals the value in each numeric column. We can use this function to find the total marks of a student, total salary of an employee in a specific period etc.
  - AVG – It gives the average value of a column, provided column has numeric value. E.g.: Average age of students present in particular class.
  - MAX – It gives the maximum value in a column. For example, highest scorer in the class can be retrieved by MAX function.
  - MIN– It gives the minimum value in a column. For example, lowest paid employee in a department can be obtained by MIN.
  - These group functions can be used with group by clause or without it.

- Having Clause - Using this clause we can add conditions to the grouped categories and filter the records. For examples, if we want to display the details of the department which has more than 100 employees.
  ```
  SELECT d.DEPATMENT_NAME, COUNT (e.DEPATMENT_ID)
  FROM EMPLOYEE e, DEPARTMENT d
  WHERE e.DEPARTMENT_ID = d.DEPARTMENT_ID
  GROUP BY e.DEPARTMENT_ID
  HAVING COUNT(e.DEPATMENT_ID)>100;
  ```
- This query filters more than 100 employees present in specific department. We cannot give this condition in the where clause as this is the result of Group by clause.

- LIKE Clause
  - The LIKE operator is used in a WHERE clause to search for a specified pattern in a column. It is used to compare a value to similar values using wildcard operators
  - There are two wildcards often used in conjunction with the LIKE operator:
    - (%) - The percent sign represents zero, one, or multiple characters
    - (_) - The underscore represents a single character
  - 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_'
    ```
  - You can combine N number of conditions using AND or OR operators. Here, XXXX could be any numeric or string value.

Examples:

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.

DCL COMMANDS

- DCL stands for Data Control Language.
- DCL is used to control user access in a database.
- This command is related to the security issues.
- Using DCL command, it allows or restricts the user from accessing data in database schema.
  DCL commands are as follows,
  1. GRANT
  2. REVOKE

GRANT COMMAND

- GRANT command gives user's access privileges to the database.
- This command allows specified users to perform specific tasks.

Syntax:
GRANT <privilege list>
ON <relation name or view name>
TO <user/role list>;

EXAMPLE:

GRANT ALL ON employee
TO ABC;
[WITH GRANT OPTION] REVOKE

COMMAND

- REVOKE command is used to cancel previously granted or denied permissions.
- This command withdraw access privileges given with the GRANT command.
- It takes back permissions from user.

Syntax:
REVOKE <privilege list>
ON <relation name or view name>
FROM <user name>;

EXAMPLE:

REVOKE UPDATE

ON employee
FROM ABC;

Difference between GRANT and REVOKE command.

GRANT	REVOKE
GRANT command allows a user to perform certain activities on the database.	REVOKE command disallows a user to perform certain activities.
It grants access privileges for database objects to other users.	It revokes access privileges for database objects previously granted to other users.
Example: GRANT privilege_name ON object_name TO {	Example: REVOKE privilege_name ON object_name FROM {

user_name|PUBLIC|role_name
}
 	
[WITH GRANT OPTION];
	user_name|PUBLIC|role_name
	}

TCL COMMANDS

- TCL stands for Transaction Control Language.
- This command is used to manage the changes made by DML statements.
- TCL allows the statements to be grouped together into logical transactions.

TCL commands are as follows:
1. COMMIT
2. SAVEPOINT
3. ROLLBACK
4. SET TRANSACTION

1. COMMIT COMMAND

- COMMIT command saves all the work done.
- It ends the current transaction and makes permanent changes during the transaction.

Syntax:
```
commit;
```

2. SAVEPOINT COMMAND

- SAVEPOINT command is used for saving all the current point in the processing of a transaction.
- It marks and saves the current point in the processing of a transaction.

  Syntax:
  ```
  SAVEPOINT <savepoint_name>
  ```

  Example:
  ```
  SAVEPOINT no_update;
  ```

- It is used to temporarily save a transaction, so that you can rollback to that point whenever necessary.

3. ROLLBACK COMMAND

- ROLLBACK command restores database to original since the last COMMIT.
- It is used to restores the database to last committed state.

Syntax:

```
ROLLBACK TO SAVEPOINT <savepoint_name>;
```

Example:

```
ROLLBACK TO SAVEPOINT no_update;
```

4. SET TRANSACTION

- SET TRANSACTION is used for placing a name on a transaction.

  Syntax:

  ```
  SET TRANSACTION [Read Write | Read Only];
  ```

- You can specify a transaction to be read only or read write.
- This command is used to initiate a database transaction.