SQL by Vaibhav Sir

Q. What is Database.

A: Any collection of related information like Phone directory, Employye Details etc called as database.

Q. What is querry?

A: Request made to database management system for spacific information. Or

A: Set of instructions given to database management system to create, retrieve, manipulate and delete information.

Q. What is SQL

A: SQL is structurrred querry language used to store, manipulate and retrieve data in database

We can use SQL for: (Create Retrieve Update Delete)

It let you access and manipulate data
Execute queries against database
Retrieve, create/ drop table, insert/update/delete records
SQL keywords are not case sensitive and semicolon is used to saperate each SQL statement.

Q. What is difference between DROP, DELETE and TRUNCATE.

A. DROP and TRUNCATE are DDL commands while DELETE is a DML Command.

If DROP Database/ DROP Table Name function is used then it will drop the databse or table also the structure of table/schema is removed out.

But in case of TRUNCATE, it is used to delete the complete table or database but structure or schema of table still remains present.

In Drop and Trancate Rollback is not Possible as DDL commands are auto-commit commands.

Delete is a DML command.

Delete statement is used to delete the complete records / specific records from table.

Rollback is possible in DML command.

Q. Different types SQL commands.

A: There are basically four types of sequal commands.

1. Data Definition Languages provides or let you work with the definition/strucrure of the table
   DDL Command are auto-commtted means permanantly saves all the changes in database.
   Rollback is not possible in DDL.
   a. CREATE is used to create a table/database.
   b. ALTER is used to alter the structure of the table. i.e. modify the field already present is table or add new fields
   c. DROP is used to delete both the structure and record stored in the table.
   d. TRUNCATE is used to delete all the rows from the table and free the space containing the table.
2. Data Manupulation Language modifies the database spacially use to work on **records/rows.**
   DML is not auto committed means roll back is possible in case of DML commands
   a. INSERT is used to insert data into the row of a table.
   b. UPDATE is used to update or modify the value of a column in the table.
   c. DELETE is used to remove one or more row from a table.
3. Data Querry Language is used to fetch the data from Database
   a. SELECT is used to retrieve certain records from the database
4. Data Control Language used to grant and take back authority from any database user.
   a. GRANT is used to give user access privileges to a database.
   b. REVOKE is used to take back permissions from the user.
5. Transaction Control Language used with DML commands only.
   a. COMMIT is used to save all the transactions to the database.
   b. ROLLBACK is used to undo transactions that have not already been saved to the database.
   c. SAVEPOINT is used to roll the transaction back to a certain point without rolling back the entire transaction.

NOTE : Once you COMMIT the transaction a ROLLBACK is not possible after commit.

Login_Detail

First Name	Surname	Mobile No	Gender	DOB
Vaibhav	Yendole	23456789	Male	04/09/1993
Akash	XYZ	3456789	Male	3456

Command Syntax

1. SELECT statement

The SELECT statement is used to select data from a database.

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

```
SELECT CustomerName, City FROM Customers;
SELECT * City FROM Customers;
```

---

2. SELECT DISTINCT statement

The SELECT DISTINCT statement is used to return only distinct (different) values.

```
SELECT DISTINCT column1, column2, ...
FROM table_name;
```

```
SELECT DISTINCT Country FROM Customers;

SELECT Count(*) AS DistinctCountries
FROM (SELECT DISTINCT Country FROM Customers);
```

---

3. WHERE Clause. (= , <, > , < > , !=, BETWEEN, LIKE, IN,UPDATE)

The WHERE clause is used to filter records.

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

```
SELECT * FROM Customers
WHERE Country='Mexico';
```

---

4. AND | OR | NOT operator.

The AND and OR operators are used to filter records based on more than one condition:

- The AND operator displays a record if all the conditions separated by AND are TRUE.
- The OR operator displays a record if any of the conditions separated by OR is TRUE.

AND Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```

OR Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2 OR condition3 ...;
```

```
SELECT column1, column2, ...                    NOT Syntax
FROM table_name
WHERE NOT condition;
```

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

SELECT * FROM Customers
WHERE Country='Germany' OR Country='Spain';

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

```
SELECT * FROM Customers
WHERE Country='Germany' AND (City='Berlin' OR City='München');

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

---

5. ORDER BY keyword

The ORDER BY keyword is used to sort the result-set in ascending or descending order. By default, Ascending

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

```
SELECT * FROM Customers
ORDER BY Country, CustomerName;(First Sort by Country then CustomerName)
```

---

6. INSERT INTO VALUES statement

The INSERT INTO statement is used to insert new records in a table.

```
INSERT INTO table_name (column1, column2, column3, ...)  Method :1
VALUES (value1, value2, value3, ...);
```

```
INSERT INTO table_name                                   Method :2
VALUES (value1, value2, value3, ...);
```

```
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');

INSERT INTO Customers
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');
```

---

7. IS NULL | IS NOT NULL operator

A field with a NULL value is a field with no value.

```
SELECT column_names
FROM table_name
WHERE column_name IS NULL; | IS NOT NULL
```

```
SELECT CustomerName, ContactName, Address
FROM Customers
WHERE Address IS NOT NULL;
```

---

8. UPDATE SET statement

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

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

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

Be careful when updating records. If you omit the WHERE clause, ALL records will be updated!

---

9. DELETE FROM statement

The DELETE statement is used to delete existing records in a table.

```
DELETE FROM table_name WHERE condition;
```

Note: Be careful when deleting records in a table! Notice the WHERE clause in the DELETE statement. The WHERE clause specifies which record(s) should be deleted. If you omit the WHERE clause, all records in the table will be deleted!

```
DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';
```

Delete All Records

Deletes all rows in the "Customers" table, without deleting the table:

```
DELETE FROM table_name;
```

---

10. SELECT TOP Clause

The `SELECT TOP` clause is used to specify the number of records to return.

```
SELECT TOP number|percent column_name(s)
FROM table_name
WHERE condition;
```

```
SELECT TOP 3 * FROM Customers;
SELECT TOP 50 PERCENT * FROM Customers;

SELECT TOP 3 * FROM Customers
WHERE Country='Germany';
```

---

11. MIN( ) | MAX( ) Function

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

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

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;


SELECT MAX(column_name)
FROM table_name
WHERE condition;
```



```
SELECT MIN(Price) AS SmallestPrice
FROM Products;
```

---

12. COUNT ( ) | AVG ( ) | SUM ( ) Function

- The `COUNT()` function returns the number of rows that matches a specified criterion.
- The `AVG()` function returns the average value of a numeric column.
- The `SUM()` function returns the total sum of a numeric column.

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

```
SELECT COUNT(ProductID)
FROM Products;

SELECT AVG(Price)
FROM Products;

SELECT SUM(Quantity)
FROM OrderDetails;
```

---

13. LIKE Operator( )

The `LIKE` operator is used in a `WHERE` clause to search for a specified pattern in a column.

There are two wildcards often used in conjunction with the `LIKE` operator:

- The percent sign (%) represents zero, one, or multiple characters
- The underscore sign (_) represents one, single character
- [charlist]% Matches more characters in charlist
- [!charlist]% Not matches characters sequence

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

```
SELECT * FROM VCTC
where Fname LIKE [AN] %;                    [AN]% = AXXXX, NXXXX
```

---

14. Wildcard character

A wildcard character is used to substitute one or more characters in a string.

Wildcard characters are used with the LIKE operator. The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

```
SELECT * FROM Customers
WHERE City LIKE 'ber%';
```

```
SELECT * FROM Customers
WHERE City LIKE '[bsp]%';            Returns City name from B, S, and P

SELECT * FROM Customers
WHERE City NOT LIKE '[bsp]%';
```

%	Represents zero or more characters	bl% finds bl, black, blue, and blob
_	Represents a single character	h_t finds hot, hat, and hit
[]	Represents any single character within the brackets	h[oa]t finds hot and hat, but not hit
^	Represents any character not in the brackets	h[^oa]t finds hit, but not hot and hat
-	Represents any single character within the specified range	c[a-b]t finds cat and cbt

---

15. IN ( ) operator / NOT IN ( ) operator

The IN operator allows you to specify multiple values in a WHERE clause.
The IN operator is a shorthand for multiple OR conditions

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (SELECT STATEMENT);
```

```
SELECT * FROM Customers
WHERE Country IN ('Germany', 'France', 'UK');

SELECT * FROM Customers
WHERE Country NOT IN ('Germany', 'France', 'UK');

SELECT * FROM Customers
WHERE Country IN (SELECT Country FROM Suppliers);
```

---

16. BETWEEN operator AND

The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.

The BETWEEN operator is inclusive: begin and end values are included.

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20;

SELECT * FROM Products
WHERE Price NOT BETWEEN 10 AND 20;

SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20
AND CategoryID NOT IN (1,2,3);

SELECT * FROM Products
WHERE ProductName BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni'
ORDER BY ProductName;

SELECT * FROM Orders
WHERE OrderDate BETWEEN #07/01/1996# AND #07/31/1996#;

SELECT * FROM Orders
WHERE OrderDate BETWEEN '1996-07-01' AND '1996-07-31';
```

---

17. ALIAS

SQL aliases are used to give a table, or a column in a table, a temporary name.

```
SELECT column_name AS alias_name
FROM table_name;
```

```
SELECT column_name(s)
FROM table_name AS alias_name;
```

```
SELECT CustomerName AS Customer, ContactName AS [Contact Person]
FROM Customers;

SELECT CustomerName, Address + ', ' + PostalCode + ' ' + City + ', ' +
Country AS Address
FROM Customers;

SELECT CustomerName, CONCAT(Address,', ',PostalCode,', ',City,',
',Country) AS Address
FROM Customers;
```

---

18. Create /Store Procedure

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

```
CREATE PROCEDURE procedure_name       //Creating Procdure
AS
sql_statement
GO;
EXEC procedure_name;              //Execution of procedure
```

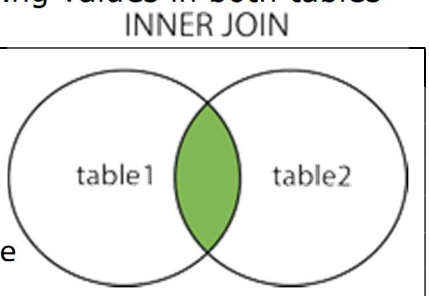
```
CREATE PROCEDURE data
AS
select * From Customers
GO;
EXEC data;                  // to execute the store procedure use Keyword EXEC
```

---

19. JOINS Clause

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

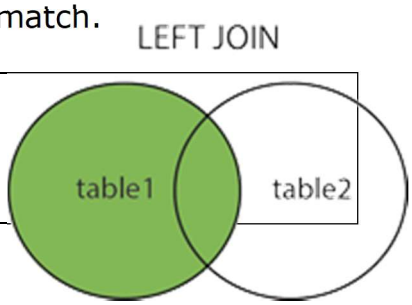
1. (INNER) JOIN: Returns records that have matching values in both tables




```
SELECT table_name.column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;

SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;

INNER JOIN FOR THREE TABLES

SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName
FROM ((Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID)
INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID);
```

2. LEFT (OUTER) JOIN: Returns all records from the left table, and the matched records from the right table

The LEFT OUTER JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side, if there is no match.

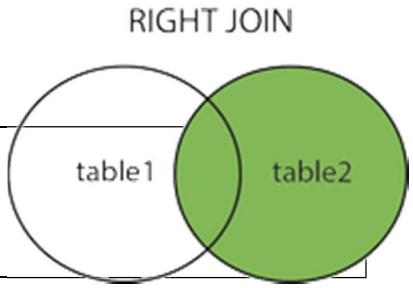



```
SELECT table_name.column_name(s)
FROM table1
LEFT OUTER JOIN table2
ON table1.column_name = table2.column_name;

SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;
```

Note: The LEFT JOIN keyword returns all records from the left table (Customers), even if there are no matches in the right table (Orders).

3. RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table

   The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.




```
SELECT table_name.column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

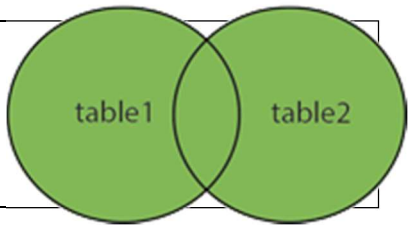
```
SELECT Orders.OrderID, Employees.LastName, Employees.FirstName
FROM Orders
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
ORDER BY Orders.OrderID;
```

In some databases RIGHT JOIN is called RIGHT OUTER JOIN.

4. FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table

   The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.

   Tip: FULL OUTER JOIN and FULL JOIN are the same.




```
SELECT table_name.column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.Customer
ID
ORDER BY Customers.CustomerName;
```

Note: The FULL OUTER JOIN keyword returns all matching records from both tables whether the other table matches or not. So, if there are rows in "Customers" that do not have matches in "Orders", or if there are rows in "Orders" that do not have matches in "Customers", those rows will be listed as well.

5. SELF JOIN keyword

A self join is a regular join, but the table is joined with itself.

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

T1 and *T2* are different table aliases for the same table.

```
SELECT A.CustomerName AS CustomerName1,
B.CustomerName AS CustomerName2, A.City
FROM Customers A, Customers B
WHERE A.CustomerID <> B.CustomerID
AND A.City = B.City
ORDER BY A.City;
```

---

20. UNION operator

The UNION operator is used to combine the result-set of two or more SELECT statements.

- Every SELECT statement within UNION must have the same number of columns
- The columns must also have similar data types
- The columns in every SELECT statement must also be in the same order

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

```
SELECT 'Customer' AS Type, ContactName, City, Country
FROM Customers
UNION
SELECT 'Supplier', ContactName, City, Country
FROM Suppliers;
```

---

21. UNION ALL operator

The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL:

Note: The column names in the result-set are usually equal to the column names in the first SELECT statement.

```
SELECT column_name(s) FROM table1
UNION ALL
SELECT column_name(s) FROM table2;
```

```
SELECT City, Country FROM Customers
WHERE Country='Germany'
UNION ALL
SELECT City, Country FROM Suppliers
WHERE Country='Germany'
ORDER BY City;
```

22. GROUP BY statement

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

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

---

23. HAVING clause

The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

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

---

24. EXISTS operator

The EXISTS operator is used to test for the existence of any record in a subquery.

The EXISTS operator returns TRUE if the subquery returns one or more records.

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS
(SELECT column_name FROM table_name WHERE condition);
```

```
SELECT SupplierName
FROM Suppliers
WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.Suppli
erID = Suppliers.supplierID AND Price < 20);
```

---

25. ANY operator

- returns a boolean value as a result
- returns TRUE if ANY of the subquery values meet the condition

ANY means that the condition will be true if the operation is true for any of the values in the range.

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ANY
  (SELECT column_name
  FROM table_name
  WHERE condition);
```

```
SELECT ProductName
FROM Products
WHERE ProductID = ANY
  (SELECT ProductID
  FROM OrderDetails
  WHERE Quantity = 10);
```

Note: The *operator* must be a standard comparison operator (=, <>, !=, >, >=, <, or <=).

---

26. ANY ALL operator

- returns a boolean value as a result
- returns TRUE if ALL of the subquery values meet the condition
- is used with SELECT, WHERE and HAVING statements

ALL means that the condition will be true only if the operation is true for all values in the range.

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ALL
  (SELECT column_name
  FROM table_name
  WHERE condition);
```

```
SELECT ProductName
FROM Products
WHERE ProductID = ALL
  (SELECT ProductID
  FROM OrderDetails
  WHERE Quantity = 10);
```

---

27. SELECT INTO statement

The `SELECT INTO` statement copies data from one table into a new table.

Copy all columns into a new table:

```
SELECT *
INTO newtable [IN externaldb]
FROM oldtable
WHERE condition;
```

```
SELECT * INTO CustomersBackup2017 IN 'Backup.mdb'
FROM Customers;
```

Copy only some columns into a new table:

```
SELECT column1, column2, column3, ...
INTO newtable [IN externaldb]
FROM oldtable
WHERE condition;
```

```
SELECT CustomerName, ContactName INTO CustomersBackup2017
FROM Customers;
```

---

28. SELECT INTO SELECT statement

The `INSERT INTO SELECT` statement copies data from one table and inserts it into another table.

The `INSERT INTO SELECT` statement requires that the data types in source and target tables matches.

Note: The existing records in the target table are unaffected.

Copy all columns from one table to another table:

```
INSERT INTO table2
SELECT * FROM table1
WHERE condition;
```

```
INSERT INTO Customers (CustomerName, City, Country)
SELECT SupplierName, City, Country FROM Suppliers;
```

Copy only some columns from one table into another table:

```
INSERT INTO table2 (column1, column2, column3,…)
SELECT column1, column2, column3, ...
FROM table1
WHERE condition;
```

```
INSERT INTO Customers (CustomerName, ContactName, Address, City,
PostalCode, Country)
SELECT SupplierName, ContactName, Address, City,
PostalCode, Country FROM Suppliers;
```

29. CASE statement

The CASE statement goes through conditions and returns a value when the first condition is met (like an if-then-else statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause.

If there is no ELSE part and no conditions are true, it returns NULL.

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    WHEN conditionN THEN resultN
    ELSE result
END;
```

```
SELECT OrderID, Quantity,
CASE
    WHEN Quantity > 30 THEN 'The quantity is greater than 30'
    WHEN Quantity = 30 THEN 'The quantity is 30'
    ELSE 'The quantity is under 30'
END AS QuantityText
FROM OrderDetails;
```

SQL Constraints

SQL constraints are used to specify rules for the data in a table

- 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 - Prevents actions that would destroy links between tables
- CHECK - Ensures that the values in a column satisfies a specific condition
- DEFAULT - Sets a default value for a column if no value is specified
- CREATE INDEX - Used to create and retrieve data from the database very quickly

---

1. CREATE DATABASE

The CREATE DATABASE statement is used to create a new SQL database.

```
CREATE DATABASE databasename;
CREATE DATABASE testDB;
```

Tip: Make sure you have admin privilege before creating any database. Once a database is created, you can check it in the list of databases with the following SQL command: SHOW DATABASES;

2. DROP DATABASE

The DROP DATABASE statement is used to drop an existing SQL database.

```
DROP DATABASE databasename;
DROP DATABASE testDB;
```

3. BACKUP DATABASE

The BACKUP DATABASE statement is used in SQL Server to create a full back up of an existing SQL database.

```
BACKUP DATABASE databasename
TO DISK = 'filepath';

BACKUP DATABASE testDB
TO DISK = 'D:\backups\testDB.bak';
```

Tip: Always back up the database to a different drive than the actual database. Then, if you get a disk crash, you will not lose your backup file along with the database.

```
BACKUP DATABASE testDB
TO DISK = 'D:\backups\testDB.bak'
WITH DIFFERENTIAL;
```

Tip: A differential back up reduces the back up time (since only the changes are backed up).

4. CREATE TABLE

The CREATE TABLE statement is used to create a new table in a database.

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....);

CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

6. DROP TABLE

The DROP TABLE statement is used to drop an existing table in a database.

```
DROP TABLE table_name;
```

Note: Be careful before dropping a table. Deleting a table will result in loss of complete information stored in the table!

```
DROP TABLE Shippers;
```

7. TRUNCATE TABLE

The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.

```
TRUNCATE TABLE table_name;
```

8. ALTER TABLE

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

```
ALTER TABLE table_name
ADD column_name datatype;

ALTER TABLE Customers
ADD Email varchar(255);

ALTER TABLE table_name
DROP COLUMN column_name;
```

SQL Commands

1. Find Even Salary/ Number

```
Select * From Employee
Where (Salary%2) = 0 or
Where (mod(Salary,2)=0;
```

2. Find ODD Salary/ Number

```
Select * From Employee
Where (Salary%2)=1 or
Where (mod(Salary,2)=1;
```

3. Find 2nd Highest Salary

```
Select * From Employee
Where Salary <(Select Max(Salary) From Employee);
```