MICROSOFT SQL SERVER

(Important keywords other then select , from and where)

Part-1

By - Krishnanand

semicolon Is required for the ending of the code

1. TOP():

```
SELECT TOP(5) artist
FROM   artists;
```

2. PERCENT:

```
SELECT TOP(5) PERCENT artist
FROM artists;
```

3. DISTINCT:

```
SELECT DISTINCT artist
FROM artists;
```

4. AS:

```
SELECT artist AS artist_name
FROM artists;
```

Ordering and Filtering:

5. ORDER BY :

```
SELECT TOP(50) , product _id
FROM shop
ORDER BY product_id ;
```

6. DESC:

```
SELECT TOP(50) , product _id
FROM shop
ORDER BY product_id DESC;
```

7. NOT EQUAL: <>

```
Select product_id
```

```
From shop
Where product_id <> 10;
```

8. BETWEEN:
```
Select product_id
From shop
Where product_id BETWEEN 10 AND 30;
```

9. NULL :
```
Select product_id
From shop
Where product_id  IS NULL;
```

10. NOT NULL:
```
Select product_id
From shop
Where product_id  IS NOT NULL;
```

11. IN():
```
SELECT song
FROM artist
WHERE artist IN('mai' ,'hun');
```

12. LIKE:
```
SELECT song
FROM artist
WHERE artist LIKE '%a';
```

Aggregating Data

13. SUM()
```
SELECT
SUM (salary) AS total_paid_expenditure
SUM(sent) AS money_lend
FROM grid;
```

14. COUNT()
```
SELECT
COUNT(employee) AS employees
FROM grid
#using Distinct will produce different results
```

15. MIN()
    SELECT
    MIN(employee) AS employees
    FROM grid
16. MAX()
17. AVG()

Strings

18. LEN():
    SELECT
    description,
    LEN(description) AS description_length
    FROM
    Grid;

19. LEFT/ RIGHT ():
    SELECT
    Description
    LEFT(description,20) AS description_20
    FROM
    Grid;

20. CHARINDEX():
    SELECT
    CHARINDEX('_' , url) AS char_location
    FROM
    Courses;

21. SUBSTING():
    SELECT
    SUBSTRING(URL,12,12) AS target_part
    FROM
    Courses;

22. REPLACE():
    SELECT
    TOP(5) REPLACE(url,'_','-') AS replacement
    FROM
    Courses;

Grouping and Having

23. PRIMARY KEYS:
    It uniquely identifies each row in the table

24. FOREIGN KEY :
    Foreign key is the one which tend to connect two tables.

25. INNER JOIN:
    It will return the value which is matched in both of the table

```
SELECT
Album_id ,
Title,
Album.artist_id,
Name AS arist_name
FROM album
INNER JOIN artist ON artist.artist_id = album.artist.artist_id
WHERE album.artist_id = 1;
```

#SYNTAX OR FORMAT IN WHICH IT SHOULD BE DONE

```
SELECT
tableA.columnX
tablebB.columnY
tableC.columnZ

FROM
TableA
INNER JOIN TableB ON tableB.foreign_key = tableB.primary_key;
```

26. LEFT JOIN:
    The LEFT JOIN keyword returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side, if there is no match.

```
SELECT
  invoiceline_id,
  unit_price,
  quantity,
  billing_state
  -- Specify the source table
FROM invoice
  -- Complete the join to the invoice table
LEFT JOIN invoiceline
ON invoiceline.invoice_id = invoice.invoice_id;
```

27. RIGHT JOIN:
    A RIGHT join will return all rows from the right hand table, plus any matches from the left hand side table.

```
-- SELECT the fully qualified album_id column from the album table
SELECT
  album_id,
  title,
  album.artist_id,
  -- SELECT the fully qualified name column from the artist table
  name as artist
FROM album
-- Perform a join to return only rows that match from both tables
INNER JOIN artist ON album.artist_id = artist.artist_id
WHERE album.album_id IN (213,214)
```

UNION & UNION ALL

28. UNION:

```
# It will exclude duplicate rows
SELECT
album_id,
title ,
artist_id
FROM album
WHERE artist_id IN(1,3)
UNION
SELECT
album_id,
title ,
artist_id
FROM album
WHERE artist_id IN(1,4,5);
```

29. UNION ALL:
    # It include duplicate rows

```
SELECT
album_id,
title ,
artist_id
FROM album
WHERE artist_id IN(1,3)
UNION ALL
SELECT
album_id,
```

```
title ,
artist_id
FROM album
WHERE artist_id IN(1,4,5);
```

FOR Both:
-> SELECT THE SAME NUMBER OF COLUMNS IN THE SAME ORDER.
-> COLOUMS SHOULD HAVE SAME DATA TYPE.
-> UNION ALL IS FASTER THEN THE UNION BECAUSE WILL NOT ELIMINATE THE DUPLICATE RESULTS.

Creator

30. CRUD OPERATION
    CREATE:
    - ➔ DATABASES , Tables or views
    - ➔ Users , permission and security groups

    READ :

    - ➔ Eg: select statements

    UPDATE:

    - ➔ Amend existing database records

    DELETE

31. CREATE:

```
CREATE TABLE test_table(
Test_date date ,
Test_name varchar(20),
Test_int int )
```

32. DATATYPES:
    - ➔ DATES: date(YYYY-MM-DD), datetime(YYYY-MM-DD hh:mm:ss)
    - ➔ Time
    - ➔ NUMERIC: int , decimal ,float , bit (1 =true , 0 = false , also accepts null values)
    - ➔ STRINGS: char , varchar , nvarchar
33. INSERT:
    (SYNTAX : INSERT INTO table_name(col_1 , col_2 , col_3) VALUES ('val 1' , 'val 2' , 'val 3'))

    OR

    INSERT SELECT

    INSERT INTO table_name (col 1 , col2)

    SELECT

Column1,

Column2,

Column3

FROM other_table

WHERE

- Condition applied

don't USE SELECT *

34. UPDATE:

```
UPDATE table
SET column = value
SET column1 = value1
WHERE
```

- Condition

35 . DELETE:

```
DELETE

FROM table

WHERE  -- condition
```

36. CLEAN ENTIRE TABLE (TRUNCATE):

```
TRUNCATE TABLE table_name
```

Declare yourself

37 . VARIABLE:

➔ It is used to avoid the repetition in the dbms

```
SELECT *
From employee
Where name = @my_artist;
```

38 . DECLARE:

DECLARE @

DECLARE @TEST_VARIABLE INT

DECLARE @MY_ARTIST VARCHAR(20)

39. SET:

```
DECLARE @TEST_INT INT
SET @TEST_INT = 5

DECLARE @MY_ARTIST VARCHAR(20)
DECLARE @MY_ALBUM VARCHAR(20);

SET @MY_ARTIST = 'AC/DC'
SET @MY_ALBUM = 'LET THERE BE ROCK' ;
SELECT –
FROM –
WHERE ARTIST = @MY_ARTIST
AND ALBUM = @MY_ALBUM
```

40. TEMPORARY TABLE:

```
SELECT
COL 1 ,
COL 2 ,
INTO #MY_TEMP_TABLE
FROM MY_EXISTING TABLE
WHERE
--- CONDITION
```

FOR REMOVING IT MANUALLY (IT WILL DELETED AUTOMATICALLY ONCE CONDITION OR SESSION ENDS)

```
DROP TABLE #MY_TEMP_TABLE
```