SQL

Data:

Data is a raw fact which describes attributes of an entity.

Attributes – Properties.

Entity – Object.

DataBase:

Database is a place/medium where we store data in a systematic and organized manner.

In database we perform some operations: which are

1. Create / Insert

2. Read / Retrieve

3. Update / Modify

4. Delete / Drop

These operations are universally called as ‘CRUE’ operations.

**DBMS:**

* DBMS is a software which is used to maintain and manage database.
* Security and authorization are the two important features provided by DBMS.
* Data is stored in the form of Files.
* To communicate with DBMS we use query language.

**RDBMS:** (Relational Database Management System)

* It is a type of a DBMS software which is used to maintain and manage the Database.
* Security and authorization are the two important features provided by RDBMS.
* Data is stored in the form Tables.
* To communicate with RDBMS we use structured query language

**RELATIONAL MODEL:**

* Relational model was introduced by E.F. CODD.
* Relational model says all data should be stored in form of table.
* Any DBMS which is integrated with Relational model or which follows the rules of E. F. CODD will be considered as RDBMS.
* DBMS **+** RM **=** RDBMS.

**TABLE:**

The Logical arrangement of rows and columns is called as Table.

**CELL:**

The Insertion of rows and column where we enter data is called as cell.

**RULES OF E. F. CODD:**

1. In RDBMS data entered into the cell must be single value.
2. In RDBMS we store everything in the form of table.
3. Details about data known as Metadata.
4. Metadata should be stored in the form of table which is called as metatable.
5. Metadata and metatable are automatically generated by the compiler.
6. According to E. F. CODD data can be stored in multiple tables. If required we can establish a connection between tables by using key attributes.
7. In RDBMS data entered into the cell or table must be valid in two ways.
8. Assigning Datatypes.
9. Assigning Constraints.

**NOTE:**

Here Datatypes are mandatory and constraints are optional.

**DATATYPES:**

IT specifies that what kind of data should be stored in particular memory location.

**TYPES OF DATATYPES:**

1. CHAR
2. VARCHAR / VARCHAR2
3. NUMBER
4. DATE
5. LARGE OBJECTS
6. CHARACTER LARGE OBJECT (CLOB)
7. BINARY LARGE OBJECT (BLOB)

**CONSTRAINTS:**

1. UNIQUE
2. NOTNULL
3. CHECK
4. PRIMARY KEY
5. FOREIGN KEY

TYPES OF SQL STATEMENTS:

Statements are used to perform the ‘CRUD’ operations in database.

DDL : Data Definition Language. (create, alter, drop, truncate)

DML : Data Manipulation Language. (insert, update, delete)

DCL : Data Control Language. (grant, revoke)

TCL : Transaction control Language. (commit, rollback, savepoint)

DQL : Data Query language. (select, projection, selection, joins)

CREATE:

To create objects in the database like tables, indexes, functions, views, procedures etc..,

SYNTAX:

CREATE TABLE TABLE\_NAME

(COLUMN1 DATATYPE, COLUMN 2 DATATYPE,…);

CREATE VIEW VIEW\_NAME AS SELECT\_FROM TABLE\_NAME;

ALTER:

Alter is used to alter the structure of the table like adding new columns, dropping existing column, changing the length of the datatype of a column and to rename the table.

SYNTAX:

ALTER TABLE TABLE\_A RENAME TO TABLE\_B.

ALTER TABLE TABLE\_NAME ADD NEW\_COLUMN DATATYPE.

ALTER TABLE TABLE\_NAME MODIFY COLUMN COLUMN\_NAME.

ALTER TABLE TABLE\_NAME DROP COLUMN COLUMN\_NAME.

DROP:

Drop is used to drop databases, objects, like tables, views, functions, procedures, indexes etc..,

SYNTAX:

DROP TABLE TABLE\_NAME OR DEOP TABLE\_NAME.

DROP VIEW VIEW\_NAME.

DROP FUNCTION FUNCTION\_NMAE.

DROP INDEX INDEX\_NAME.

TRUNCATE:

Truncate it is used to delete all rows from the table and we can’t rollback the deleted records.

SYNTAX:

TRUNCATE TABLE TABLE\_NAME.

INSERT:

To insert data into a table by using insert we can:

Insert single record manually

Insert multiple records manually

Insert bulkdata from other processes

Insert data from one table to another table

SYNTAX:

INSERT INTO TABLE\_NAME (columns)

VALUES(….);

INSERT INTO TABLE\_NAME VALUES(.,.,.,),(.,.,.,);

INSERT INTO TABLE\_A (columns) SELECT (columns) FROM TABLE\_B.

Example: insert into employees\_data2 (emp\_id,first\_name,last\_name,sal) select employee\_id,first\_name,last\_name,salary from employees\_data;

UPDATE:

To modify data from the table .we use **SET** keyword for updating data.

by using update we can:

Update one or more columns data

Update one or more records based on condition

Update entire table.

SYNTAX:

UPDATE TABLE\_NAME SET COLUMN1 =…. COLUMN 2=….;

UPDATE TABLE\_NAME SET COLUMN1 WHERE CONDITION.

DELETE:

To remove records from table by using delete we can:

Delete one or more records based on condition

Delete all records.

SYNTAX:

DELETE FROM TABLE\_NAME.

DELETE FROM TABLE\_NAME WHERE CONDITION.

IMPORTANT QUESTION:

WHAT IS THE DIFFERENCE BETWEEN DELETE AND TRUNCATE ?

We can delete one or more or all records from a table by using delete statement based on conditions, and we can rollback the delete data if we needed. delete is a DML statement. truncate will delete all the records and the deleted data can’t be rolled back. it is a DDL statement.

COMMIT:

It is used to permanently save any transaction into the database.

SYNTAX:

COMMIT

ROLLBACK:

It is used to rollback the transaction up to a point we can rollback the transaction made after the point.

SYNTAX:

ROLLBACK

SAVEPOINT:

It is used to save transaction up to a point and we can rollback the transactions made after the point .

SYNTAX:

SAVE TRANSACTION SP1;

Transactions….

ROLLBACK TRANSACTIONS SP1;

DQL (Data Query Language):

. Select is the only DQL command we use to retrieve data from databases.

. Select is very important command that we use to fetch data from tables and views.

. Select command plays key role in data analysis.

SYNTAX:

SELECT \*/COLUMN\_LIST **FROM**  TABLE /VIEW\_NAME.

**QUERIES:**

select \* from table\_name;

select employee\_id, first\_name, salary as mon\_sal, salary\*12 as year\_sal FROM table\_name;

select first\_name, last\_name, first\_name || last\_name as full\_name FROM table\_name;

select first\_name, last\_name, first\_name || ' ' || last\_name as full\_name FROM table\_name;

select 20+20 from dual;

select 20+20 from dummy;

WHERE CLAUSE:

1. We use **where clause** to filter data from tables / views.
2. We can mention one or more conditions in where clause based on our requirements.
3. The **select** statement displays all the records satisfying the conditions mentioned in **where.**

SYNTAX:

SELECT \* / COLUMN\_LIST **FROM** TABLE / VIEW\_NAME **WHERE**  CONDITION(S).

**NOTE:**

**SQL**  keywords are case in-sensitive, we can use select or SELECT, WHERE or where . But data we are operating is case case-sensitive, so ‘a’ is different from ‘A’. in data.

ORDER BY CLAUSE:

1. To order the data in ascending order or descending order .
2. Default is ascending order, have to specify ‘desc’ for descending order.
3. Can be applied on any data type (Numeric, Strings, Date).
4. Can be applied on multiple columns.

SYNTAX:

Select \*/column\_list FROM table /view\_name ORDER BY Col2;

Select \*/column\_list FROM table /view\_name ORDER BY Col3 desc;

Select \*/column\_list FROM table /view\_name ORDER BY Col1 desc, Col2;

**Example Queries:**

**WHERE**:

1. Select \* from table\_name WHERE dept\_id=20;

2. Select employee\_id, first\_name, salary FROM table\_name WHERE salary >= 10000;

3. Select employee\_id, first\_name, salary as mon\_sal, salary\*12 as year\_sal FROM table\_name WHERE salary\*12 >= 200000;

4. employee\_id, first\_name, hire\_data FROM table\_name WHERE hire\_data >= TO\_DATE (‘2000-01-01’, ‘yyyy-mm-dd);

5. select employee\_id, department\_id, first\_name, salary FROM table\_name WHERE dept\_id=50 AND salary >= 5000;

**BETWEEN**:

1. select employee\_id, first\_name, salary FROM table\_name WHERE salary >= 8000 and salary <= 10000;
2. select employee\_id, first\_name, salary FROM table\_name WHERE salary between 8000 and 10000;
3. select employee\_id, first\_name, salary FROM table\_name WHERE salary not between 8000 and 10000;

**ORDER BY:**

1. select \* from table\_name ORDER BY first\_name;
2. select emp\_id, first\_name, salary FROM table\_name ORDER BY salary desc;
3. select emp\_id, first\_name, hire\_date, FROM table\_name ORDER BY hire\_date;
4. select dept\_id, first\_name, last\_name, salary FROM table\_name ORDER BY dept\_id, salary desc;

STRING MATCHING:

**Like:** -To match Strings

**Not like:** -Opposite match of Strings

**Wildcard Operators:** ­-  **‘**\_’ **and ‘%’**

**‘**\_**’** ---- To match exactly single character

**‘%’** ---- To match zero or more characters

**Example Queries**:

1. select employee\_id, first\_name, last\_name FROM table\_name WHERE first\_name LIKE ‘A%’;

2. select employee\_id, first\_name, last\_name FROM table\_name WHERE first\_name NOT LIKE ‘A%’;

3. select employee\_id, first\_name, last\_name FROM table\_name WHERE first\_name LIKE ‘\_a%’;

4. select employee\_id, first\_name, last\_name FROM table\_name WHERE first\_name LIKE ‘%a’;

5. select employee\_id, first\_name, last\_name FROM table\_name WHERE first\_name LIKE ‘A % a’;

6. select employee\_id, first\_name, last\_name FROM table\_name WHERE first\_name LIKE ‘A%’ and last\_name LIKE ‘A%’;

7. select employee\_id, first\_name, last\_name FROM table\_name WHERE first\_name LIKE ‘A\_ \_ \_ \_’;

8. select employee\_id, first\_name, last\_name FROM table\_name WHERE first\_name LIKE ‘% %’;

9. select employee\_id, first\_name, last\_name FROM table\_name WHERE last\_name LIKE ‘\_’;

10. select employee\_id, first\_name, last\_name FROM table\_name WHERE last\_name LIKE ‘\_’ or last\_name LIKE ‘\_ \_’;

PREDEFINED FUNCTIONS:

1. Numeric Functions
2. String Functions
3. Date and Time Functions
4. Conversion Functions
5. Aggregate Functions
6. Null handling Functions
7. Window Functions

**Numeric Functions:**

1. ABS -- Returns the absolute(positive) value of numeric expression
2. MOD -- Equivalent to the % which returns remainder
3. SQRT -- Returns the square-root of a non-negative number of expression
4. SQUARE – Returns the Square of a numeric expression
5. POWER(x, y) – Returns x to the power of y value
6. CELL -- Returns the nearest equal or larger integer
7. FLOOR – Returns the nearest equal or large integer
8. ROUND – Returns the nearest integer based on the decimal part
9. SIGN – Returns sign of its argument(-1 for negative, 1 for the positive, 0 for 0)
10. LOG – Returns the logarithm of a number or expression
11. LN – Returns the natural logarithm of a number or expression
12. EXP (x) -- Returns Euler number (e – 2.718282) to the power of x
13. FACTORIAL – Returns factorial of a number or expression

**Example Queries**:

1. Select abs(1.25), abs(-4.71), abs(-981), abs(0) from dual;

2. Select mod(25,4), mod(-39,5), mod(1234,7) from dummy;

3. Select sqrt(225), sqrt(12), power(5,4);

4. Select ceil(2.3), ceil(-4.8), ceil(0);

5. Select floor(2.3), floor(-4.8), floor(0);

6. Select round(2.3), round(2.8), round(-4.8), round(2.5);

7. Select sign(9), sign(-9), sign(0);

8. Select log(10,10), log(10,100), ln(10);

9. Select exp(2);

10. Select factorial(6);

//Fetch all employee details with even number employee.

11. Select \* from table\_name WHERE mod(employee\_id,2)=0;

//Divide the input data into 4 sets.

12. Select \* from table\_name WHERE mod(employee-id,4)=0;

13. Select \* from table\_name WHERE mod(employee-id,4)=1;

14. Select \* from table\_name WHERE mod(employee-id,4)=2;

15. Select \* from table\_name WHERE mod(employee-id,4)=03;

**String Functions:**

1. LENGTH(LEN) – To calculate length of given string
2. CONCAT – To combine two or more strings
3. TRIM – To remove spaces on both left and right sides of the given strings
4. LTRIM – To remove spaces or special characters on the left side of the given string
5. RTRIM – To remove spaces or special characters on the right side of the given string
6. LPAD – To add spaces or characters on the left side of the given string
7. RPAD -- To add spaces or characters on the right side of the given string
8. SUBSTR(SUBSTRING) – To get a part of the given string
9. REVERSE – Returns reverse of given string
10. LOWER – Converts given string to lower case
11. UPPER -- Converts given string to upper case
12. INITCAP – Converts first letter of each word in the given string to upper case
13. REPLACE – To replace one or more characters / words with other words

**Example Queries**:

**//** Length

1. 1. Select employee\_id, length(employee\_id), first\_name, length(first\_name), last\_name, length(last\_name) FROM table\_name;
2. //Derive the full name of employees
3. 2. Select first\_name, last\_name, concat (first\_name, ' ' , last\_name) as full\_name FROM table\_name;
4. 3. Select first\_name, last\_name, first\_name | | ' ' | | last\_name) as full\_name FROM table\_name;
5. 4. Select first\_name, last\_name, concat(first\_name, ' \_ ' , last\_name) as full\_name FROM table\_name;
6. // Trim
7. 5. Select trim(‘ I am Learning sql ’);
8. 6. Select length(‘ I am Learning sql ‘);
9. //Ltrim, Rtrim
10. 7. Select ltrim(‘ I am Learning sql ‘), rtrim (‘ I am Learning sql ‘);
11. 8. Select length(ltrim(‘ I am Learning sql ‘)), length (rtrim(‘ I am Learning sql ‘);
12. 9. Select rtrim(ltrim(‘ I am Learning sql ‘));
13. //Trim leading and trailing special characters from a string.
14. 10. Select trim(‘#123#675#912##’, ‘#’)
15. ;

**Date and Time Functions:**

1. CURRENT \_DATE() – Returns today’s date in server time zone
2. CURRENT\_TIME() – Returns current time in server time zone
3. CURRENT\_TIMESTAMP() – Returns current timestamp in server time zone
4. YEAR(date or ts) – Returns year from given date or timestamp
5. MONTH(date or ts) – Returns month from given date or timestamp(1 to 12)
6. DAY(date or ts) – Returns day of the month from given date or timestamp(1-31)
7. WEEK(date or ts) – Returns week number (1-53) of the year from given date or ts
8. TO\_DATE(expr, format) – converts given expr of certain format to date
9. TO\_TIMESTAMP(expr, format) – converts given expr of certain format to timestamp
10. DATEDIFF – To calculate difference between 2 dates in years, months, days, hours etc,..
11. TIMESTAMPDIFF – To add years, months, days, hours etc, to a Date
12. DATEADD – To add years, months, days, hours, etc, to a date
13. TIMESTAMPADD – To add years, months, days, hours etc. to a timestamp
14. MONTHS\_BETWEEN – To calculate the months between two dates or timestamps.

**Example Queries**:

1. Select current\_date, current\_time, current\_timestamp;

2. Select sysdate(), current\_timestamp;

3. Alter account SET timezone = ‘UTC’;

4. Select sysdate(), current\_timestamp;

5. Select current\_timestamp, year(current\_timestamp) as year, month(current\_timestamp) as mon, day(current\_timestamp) as day, week(current\_timestamp) as week;

6. Select to\_date(‘2023-05-12’ , ‘YYYY-MM-DDD’);

7. Select to\_date(‘2023/05/12’, ‘YYYY/MM/DD’);

8. Select to\_date(‘12/may/2023’, ‘MM/DD/YYYY’);

9. Select to\_dat(‘5/12/23’, ‘MM/DD/YY’);

10. Select to\_date(’23-12-5’, ‘YY-DD-MM’);

11. Select to\_date(‘20230512’,’YYYYMMDD’);

**Conversion Functions:**

1. TO\_CHAR – To convert a decimal or date expression to char type
2. TO\_DECIMAL OR TO\_NUMBER – To convert a string with only numeric characters to decimal, If string has non-numeric characters, conversion will fail
3. TO\_DATE(expr, format) – Converts given expr of certain format to Timestamp
4. CAST – To convert any datatype to other datatype

// To char

1. Select to\_char(12.34);
2. Select to\_char(current\_date);
3. Select to\_char(current\_timestamp);
4. Select to\_char(‘5/17/2023’);
5. Select to\_char(‘5/17/2023’::DATE);

// To\_decimal ---- To\_Number --- Both are Same

1. Select to\_decimal (‘000056’);
2. Select to\_decimal (’12.34’);
3. Select to\_decimal(’12.34’, 10,2);
4. Select to\_decimal(‘678a9’);

// to\_date – to convert any format of input dates to one date format

1. Select to\_date(‘2023-05-12’, ‘YYYY-MM-DD’);
2. Select to\_date(‘12/May/2023’, ‘DD/Mon/YYYY’);
3. Select to\_date(’23-12-5’, ‘YY-DD-MM’);
4. Select to\_date(‘20230512’, ‘YYYYMMDD’);

// to\_timestamp

1. Select to\_timestamp(‘13/05/2023 21:32:53’, ‘DD/MM/YYYY HH24:MI:SS’);
2. Select to\_timestamp(‘May-13-23 21:32:53’, ‘Mon-DD-YY HH24:MI:SS’);

**AGGERGATE FUNCTIONS:**

1. Aggregate functions are used to calculate summaries after grouping the dats.
2. Summaries like count, Minimum, Maximum, Sum, Average, Mode, Median.
3. Aggregate function performs calculation on multiple values of a colimn and returns a single value.
4. If we want to use Aggregate functions over some groups of data, we have to group the data first based on some column(s) or expression(s).
5. We use GROUP BY caluse to group the data.

COUNT – To get the number of records from a table or view or from a group

SUM – To get the sum of all values of a column

AVG – To get average of all values of a column

MAX – To get the biggest value of a column

MIN – To get smallest value of a column

MODE – To get the most frequent value of values of a column

MEDIAN – To get middle value of column

**Note:** Expect **COUNT (\*)**  none of the other aggregate functions consider nulls.

SYNTAX:

Select column/expr, aggr\_fun FROM table\_name GROUP BY column/expr;

HAVING**:**

Having clause is used to filter out the data from groups based on conditions, it is used after GROUP BY clause.

SYNTAX:

Select column/expr, aggr\_fun FROM table\_name GROUP BY column/expr HAVING conditions;

**IMP INTERVIEW QUESTIONS:**

1. What is the difference between WHERE and HAVING ?

Ans: Where is used to filter out the data from a table or view based on conditions, but Having is used to filter out the data from groups of records based on specified conditions, Having is used along with GROUP BY clause.

1. Can you use both WHERE and HAVING in same SQL statement?

Ans: Yes, we can use.

1. What is the order of execution of WHERE and HAVING in the query?

Ans: SELECT ---- FROM WHERE ---- GROUP BY ---- HAVING---

**Example Queries**:

// Simple Aggregate functions:

SELECT COUNT(\*) FROM table\_name;---107

SELECT SUM(salary), AVG(salary) FROM table\_name;

SELECT ROUND (SUM(salary), cast(AVG(salary) as NUMBER(10,2))) AS AVG\_SAL FROM table\_name;

SELECT MIN(salary), MAX(salary) FROM table\_name;

SELECT MODE(salary), MEDIAN(SALARY) FROM table\_name;

//GROUP BY

// Dept wise employee count

SELECT dept\_id, COUNT(\*) FROM table\_name GROUP BY dept\_id;

// Dept wise sum and Average salaries

SELECT dept\_id, SUM(salary), cast(AVG(salary) as NUMBER(10,2)) AVG\_SAL FROM table\_name GROUP BY dept\_id;

//Dept wise highest salary

SELECT YEAR(HIRE\_DATE), count(\*) FROM table\_name GROUP\_BY YEAR(hire\_date);

// Get count of employees hired in each year Dept wise

SELECT YEAR (hire\_date), dept\_id, count(\*) FROM table\_name GROUP BY YEAR (hire\_date), dept\_id;

**Null handling Functions:**

What is Null?

1. A column with NULL value means that column has no value
2. A NULL value neither Zero(0) nor a blank space
3. Length of a NULL value is NULL where length of a blank value is 0.
4. We can define nullability of a column in the table definition, called NOT NULL constraint.
5. If we define any column as NOT NULLABLE then it will not accept nulls, if you try to load nulls onto that, the query will fail.
6. NULL never matches with another NULL.
7. We should be very careful with NULLS.

IFNULL(col1/expr1,col2/expr2)—If col1 is null then returns col2 otherwise returns col1.

NVL(col1/expr1,col2/expr2)— If col1 is null then returns col2 otherwise returns col1.

NVL2(col1/expr1,col2/expr2, col3/expr3)—If col1 is not null then returns col2 otherwise returns col3.

COALESCE(<expr1>,<expr2>[….., <exprN>], default)—returns first not-null expression.

**Example Queries**:

// Create sample DataSet

CREATE TABLE EMPLOYEES(EMP\_ID int NOT NULL,FIRST\_NAME VARCHAR(10),MIDDLE\_NAME VARCHAR(10),LAST\_NAME VARCHAR(10),AGE int, SALARY FLOAT,PRIMARY KEY(EMP\_ID));

INSERT INTO EMPLOYEES VALUES

(101,'ARUN','KUMAR','ANNA',28,35000),

(102,'KUSUMA','NULL','NAVEEN',24,55000),

(103,'NULL','UMA','DEVI',26,45000),

(104,'GEETHA','REDDY','NULL',25,41000),

(105,'VARUN','NULL','NULL',31,65000);

// Derive full name of employees

SELECT EMP\_ID, FIRST\_NAME ||' '|| MIDDLE\_NAME|| ' '|| LAST\_NAME AS FULL\_NAME FROM EMPLOYEES;

// IFNULL

SELECT EMP\_ID, TRIM(IFNULL(FIRST\_NAME,' ') || ' ' || IFNULL(MIDDLE\_NAME,' ')|| ' ' ||IFNULL(LAST\_NAME,' ')AS FULL\_NAME FROM EMPLOYEES;

//NVL

SELECT EMP\_ID,AGE,SALARY FROM EMPLOYEES;

SELECT EMP\_ID,AGE,NVL(SALARY,0) FROM EMPLOYEES;

//COALESCE

INSERT INTO EMPLOYEES VALUES

(106,NULL,NULL,NULL,27,24000);

//Get first\_name of employees, if not get last\_name of employees, if not get last\_name of employees if not name available load as N/A.

SELECT EMP\_ID, NVL(FIRST\_NAME, NVL(LAST\_NAME, NVL(MIDDLE\_NAME, 'NA'))) AS NAME, AGE, NVL(SALARY,0) FROM EMPLOYEES;

SELECT EMP\_ID, COALESCE(FIRST\_NAME,LAST\_NAME,MIDDLE\_NAME, 'NA')AS NAME,AGE,NVL(SALARY,0) FROM EMPLOYEES;

**Window Functions:**

A window is a group of related rows. A window can consist of one, or multiple rows. A window function is any function that operates over a window of rows.

**RANK** – Returns rank over a group of values, skips the series in case of duplicates

**DENSE** **RANK** – Returns rank over a group of values, doesn’t skips the series in case of duplicates

**ROW\_NUMBER** – Returns a unique row number for number for each row within a group of values

**LEAD –** To get the next row information

**LAG** -- To get the next row information

**FIRST\_VALUE** – Returns the first value within an ordered group of values

**LAST\_VALUE** -- Returns the last value within an ordered group of values

**NTH\_VALUE --** Returns the nth value within an ordered group of values

**RANK:**

1. Returns the rank of a value within an ordered group of values
2. The rank value starts at 1 and continues up sequentially
3. If two values are the same, they have the same rank but skips the next sequence number
4. Used to assign rank numbers, find the top values etc.

**SYNTAX:**

**RANK() OVER([partition by col/expr] ORDER BY col/expr [asc/desc])**

**DENSE** **RANK:**

1. Returns the rank of a value within a group of values, without gaps in the ranks.
2. The rank value starts at 1 and continues up sequentially
3. If two values are the same, they have the same rank, but doesn’t skips the next sequence number
4. Used to assign rank numbers, find the top values etc.

**SYNTAX:**

**DENSE\_RANK() ([partition by col/expr] order by col/expr[asc/desc])**

**ROW\_NUMBER:**

1. Returns a unique row number for each row within a window partition
2. The row number starts at 1 and continues up sequentially
3. If two values are the same, assigns row number randomly
4. Used to assign a sequence number irrespective of duplicates

**SYNTAX:**

**ROW\_NUMBER() OVER ([partition by col/expr] order by col/expr [asc/desc])**

**LEAD:**

Can fetch the value for a particular column from subsequent(next) row in the same table/group after sorting in some order without using a self join.

**SYNTAX:**

**LAG (col/expr, [, <offset>, <default>])**

**OVER ([PARTITION BY <col/expr>] ORDER BY <col/expr> [{ASC | DESC}])**

**LAG:**

Can fetch the value for a particular column from previous row in the same table/group after sorting in some order, without using a self join.

**SYNTAX:**

**LAG (col/expr, [, <offset>, <default>])**

**OVER ([PARTITION BY <col/expr>] ORDER BY <col/expr> [{ASC | DESC}])**

**FIRST\_VALUE:**

Returns the first value within an ordered group of values.

**SYNTAX:**

FIRST\_VALUE(<col/expr>)

OVER ([PARTITION BY <col/expr>] ORDER BY <col/expr> [{ASC | DESC}]

**LAST\_VALUE:**

Returns the last value within an ordered group of values.

**SYNTAX:**

LAST\_VALUE(<col/expr>)

OVER ([PARTITION BY <col/expr>] ORDER BY <col/expr> [{ASC | DESC}]

**NTH\_VALUE:**

Returns the NTH value within an ordered group of values.

**SYNTAX:**

LAST\_VALUE(<col/expr>, n)

OVER ([PARTITION BY <col/expr>] ORDER BY <col/expr> [{ASC | DESC}]

**Example Queries**:

Create table table\_name(emp\_id int, year int, revenue number(10,2));

Insert into table\_name values(1, 2010, 1000),(2, 2010, 10000), (3, 2011, 800), (1, 2011, 1500), (2, 2011, 12500), (2, 2012, 1200), (1, 2012, 500), (1, 2012, 15000), (3, 2013, 1100), (1, 2013, 750), (2, 2013, 20000);

//LAG and LEAD

// Difference in sales with prev year

SELECT emp\_id, year, revenue, LAG(revenue, 1, 0) OVER (PARTITION BY emp\_id ORDER BY year) as rev\_prev,

revenue – rev\_prev AS diff\_in\_rev

FROM table\_name ORDER BY emp\_id, year;

(or)

Select emp\_id, year, revenue, LAG(revenue, 1, 0) OVER (PARTITION BY emp\_id ORDER BY year) as rev\_prev,

Revenue – LAG(revenue, 1, 0) OVER (PARTITION BY emp\_id ORDER BY year) AS diff\_in\_rev FROM table\_name ORDER BY emp\_id, year;