Structured Query Language(SQL)

Data Definition Language - Data Manipulation Language - Basic SQL Queries – Views

ADVANTAGES:

1. It is an English-like language, easy-to-use yet powerful database query language.
2. It is a non-procedural language (i.e., user needs only to specify the system what data to get from database, not how to get the information requested by the user.)
3. Standard. It means it is supported by all major RDBMS. Thus skills learned on one platform may be applied to another platform.
4. Portable: Programs written in SQL are portable, i.e., programs written on one platform can be transferred to another platform and they work fine.

TYPES OF SQL:

- DDL (Data Definition Language)
- DML (Data Manipulation Language)
- DCL (Data Control Language).

DATA DEFINITION LANGUAGE (DDL):

There are three main commands in DDL.

1.CREATE

2.ALTER

3.DROP

4. View

5.Truncate

6. Rename

1.CREATE COMMAND:

Used for Creation of a table

Syntax: CREATE TABLE table-name (Column_name1 datatype(size) constraint,

Column_name2 datatype(size) constraint,

…

);

Basic components necessary to create a table:

A table must have the following components:

- The key words CREATE TABLE
- A name for the table you want to create
- Opening bracket
- At least one column with its data type.
- Optional size of the data type and constraints on the column
- Closing bracket
- SQL statement terminator –a semicolon

Syntactical and naming conventions to be followed when creating a table:

- Each column must be separated by a comma (,) except the last column.
- Table and column names must start with an alphabet, but they can include alphabets, numerals, and underscores.
- Blank spaces cannot be used in a table name. In order to separate the table name, the only character allowed is an underscore.
- Table name cannot exceed 30 characters in length
- Table name must be unique in the Schema.
- Column names in a table must be unique.

Constraints:

The purpose of constraints is to enforce some checks and balances to maintain the integrity of the database e.g. primary key, foreign key etc. These constraints will also reduce the work at the application level by letting the database handle the work of maintaining the integrity of the data. Constraints can be specified either at the column level (Column constraint) or at the table level (Table constraint).

Note: Constraints on multiple columns e.g. composite primary keys are defined using table constraints. Single column constraints can be defined using either table constraints or column constraints.

Column constraint:

```
CREATE TABLE employee
  (    empno          NUMBER (4)        PRIMARY KEY,
ename          VARCHAR2(10),
job            CHAR(9)           NOT NULL,
mgr_id         NUMBER(4)     REFERENCES employee(empno),
hiredate       DATE,
salary         NUMBER(7,2),
commission     NUMBER(7,2),
deptno         NUMBER(2)         REFERENCES DEPT(DEPTNO)
```

);

Table constraint:

CREATE TABLE employee
(empno NUMBER(4),
ename VARCHAR2(10),
…
deptno NUMBER(2) *NOT NULL*, *PRIMARY KEY* (empno, ename)
);

CREATE TABLE employee
(empno NUMBER(4),
ename VARCHAR2(10),
…
deptnoNUMBER(2) *NOT NULL*,*FOREIGN KEY*(deptno) *REFERENCES*dept (deptno)
);

Note: The FOREIGN KEY reserved word should be used only for table constraints.

CONSTRAINT clause:

Constraints can be named by users using the CONSTRAINT clause as shown below.

CREATE TABLE employee
(empno NUMBER(4),
ename VARCHAR2(10),
…
deptno NUMBER(2) *NOT NULL*,CON*STRAINT*pk_emp*PRIMARY KE*(empno, ename)
);

The primary key constraint is named as pk_emp in the above example. In the following example, the foreign key constraint is named fk_deptno.

CREATE TABLE employee
(empno NUMBER(4),
ename VARCHAR2(10),
…
deptno NUMBER(2) *NOT NULL*,
*CONSTRAINT*fk_deptno*FOREIGN KEY* (deptno) *REFERENCES*dept (deptno)
);

2.ALTER COMMAND:

Tables can be altered in one of two ways: by changing a column's definition, or by adding a column.

Adding a column:
Syntax: ALTER TABLE table_name ADD ([column] / [constraint]);
Ex: ALTER TABLE employee ADD (deptnoNUMBER(2));

Rules for adding a column:
- When adding a column, NOT NULL cannot be specified with the column (if some data already exists in table).
- NOT NULL column can be added in three steps:
  1. Add the column without NOT NULL specified
  2. Fill every row in that column with data
  3. Modify the column to be NOT NULL

Modifying a column:

Syntax: ALTER TABLE table_name MODIFY ([column] / [constraint]);

Ex: ALTER TABLE employee MODIFY (last_name VARCHAR2(15),
first_nameVARCHAR2(20) NOT NULL);

Rules to modify a column:
- Increase a character column's width at any time
- Increase the number of digits in a NUMBER columns at any time
- Increase or decrease the number of decimal places in a NUMBER column at any time

3.DROP COMMAND:
Used to drop tables when they are no longer needed.
Syntax: DROP TABLE table_name;

Ex: DROP TABLE employee;

SQL> drop table Studentt;

Table dropped.

SQL>descStudentt;
ERROR:
ORA-04043: object Studentt does not exist

4.TRUNCATE COMMAND :

Syntax:

truncate table <table_name>;

Description:

The details in the table are deleted but the table structure remains.

Example:

SQL> truncate table Student;

Table truncated.

SQL>desc Student;

Name	Null?	Type
REGNO		NUMBER(8)
NAME		VARCHAR2(15)
DEPT		VARCHAR2(20)
YEAR		NUMBER(4)

5.RENAME COMMAND :

Syntax:

rename<old_table_name> to <new_table_name>;

Description:

The old table name is replaced with the new table name.

Example

SQL> rename Student to Studentt;

Table renamed.

SQL>desc Student;
ERROR:
ORA-04043: object Student does not exist

SQL>descStudentt;

Name	Null?	Type

REGNO	NUMBER(8)
NAME	VARCHAR2(15)
DEPT	VARCHAR2(20)
YEAR	NUMBER(4)

6.CREATE VIEW COMMAND :

Syntax:

create view <view_name> as select <field_name> from <table_name> where <condition>;

Description:

A view is named, derived, virtual table. A view takes the output of a query and treats it as a table; therefore, a view can be thought of as a "stored query "or a "virtual table". The tables upon which a view is based are called base tables.

Example:

SQL> create view Studentview as select * from Student;

View created.

SQL>descStudentview;

Name	Null?	Type
REGNO		NUMBER(8)
NAME		VARCHAR2(15)
DEPT		VARCHAR2(20)
YEAR		NUMBER(4)

DROP VIEW COMMAND :

Syntax:

drop view <view_name;

Description:

A View can be dropped (deleted) by using a drop view command.

Example:

SQL> drop view Studentview;

View dropped.

7.CREATE DUPLICATE TABLE :

SQL> create table Stud as select * from Student;

Table created.

SQL>desc Stud;

Name	Null?	Type
REGNO		NUMBER(8)
NAME		VARCHAR2(15)
DEPT		VARCHAR2(20)
YEAR		NUMBER(4)

DATA MANIPULATION LANGUAGE (DML):

There are four commands in DML.

1.SELECT (This actually doesn't belong to DML, but is considered a part of it)

2.UPDATE

3.INSERT

4.DELETE

INSERT COMMAND :

Syntax:

insert into <table_name> values(Column_name);

Description:

The 'insert into' command insert the values in the specified table .In the insert into SQL sentence the columns and values have a one to one relationship (i.e) the first value described into the first column, the second value described being inserted into the second column and so on.

Example:

SQL> insert into Student values(&regno,'&name','&city','&dept');
Enter value for regno: 100
Enter value for name: A
Enter value for city: Cuddalore
Enter value for dept: IT
old 1: insert into Student values(&regno,'&name','&city','&dept')

new 1: insert into Student values(100,'A','Cuddalore','IT')

1 row created.
(Or)

SQL>insert into student values(1001,’Radha’,’Chennai’,’IT’);

DELETE COMMAND :

Syntax:

delete from <table_name> [where <Condition>];

Description:

The delete in SQL is used to remove rows from table. To remove
All the rows from a table.
(OR)
A select set of rows from a table.

Example:

SQL> delete from Student where regno=101;

1 row deleted.

UPDATE COMMAND :

Syntax:

update<table_name> set <field_name>=<Expression/Values> [where <Condition>];

Description:

The update command is used to change or modify data values in a table. To update
All the rows from a table.
(OR)
A select set of rows from a table.

Example:

SQL> update Student set dept='CSE' where regno=101;

1 row updated.

SELECT COMMAND:

SELECT command is used to retrieve the information from a table or tables.
Format: SELECT *column1, column2…*
FROM *table_name*
WHERE *conditions*;

SELECT clause

The SELECT clause is used to specify which columns are to be selected. In the following statement '*' retrieves all the columns from the table employee.

```
Ex: SELECT *
    FROM employee;
```

FROM clause

FROM is used to specify the names of the table or tables from which the data is retrieved.

WHERE clause:

The WHERE clause is used to specify what qualifiers are to be imposed on the data to be retrieved. This is known as SELECTION. A table that is built from columns from one or more tables is called a PROJECTION, or a RESULT TABLE. The following statement retrieves all the records whose department_id is 13 and reports only those columns, which are in the SELECT clause.

```
Ex: SELECT employee_id, last_name
    FROM employee
   WHERE department_id = 13;
```

DISTINCT keyword:

The following statement retrieves all the manager_ids with no duplicates.

```
Ex: SELECT DISTINCT manager_id
   FROM employee;
```

ORDER BY clause:

The ORDER BY clause is used to sort the selected data in a specific order. By default, records are sorted in ascending order. The key word DESC is used to sort in the descending order (ASC is used for ascending).

```
Ex: SELECT first_name
        FROM employee
        ORDER BY first_name;
```

SINGLE VALUE CONDITIONS:

RELATIONAL OPERATORS:

Logical operators such as '=', '<', '>', '<=', '>=', '!=' or '^=' or '<>' are used to compare a column or expression with a single value.

Ex: SELECT last_name, first_name, job_id
FROM employee
WHERE first_name='JOHN'

WILD CARD (%) & POSITION MARKER (_):

LIKE performs pattern matching. An underscore(_) represents one character. A percent sign (%) represents any number of characters. The following statement reports all the records that have 'JO' in last_name column.

Ex: SELECT first_name
FROM employee
WHERE last_name LIKE '%JO%'.

NULL checking:

NULL tests to see if data exists in a column for a row. If the column is completely empty, it is said to be NULL, and if it is non-empty, it is said to be NOT NULL. The keyword 'IS' must be used to check if a value is NULL or NOT NULL. The following statement returns all the records where data does not exist in manager_id column. (NULL is generally used to represent a value which is not known, NULL is not equal to any value)

Ex: SELECT first_name, last_name
FROM employee
WHERE manager_id IS NULL;

The following statement returns all the records where data exists in commission column.

Ex: SELECT first_name, last_name
FROM employee
WHERE commission IS NOT NULL;

MULTI-VALUE CONDITIONS:

IN:

IN checks if the expression is equal to any member of the given list of values.IN works with VARCHAR2, CHAR, DATE, and NUMBER data types. The following statement retrieves all records whose department_id is 10,20 or 30.

Ex: SELECT first_name

FROM employee
WHERE department_id IN (10, 20, 30);

NOT IN:

NOT IN means the expression is not equal to any member of the given list of values. NOT IN works with VARCHAR2, CHAR, DATE, and NUMBER data types. The following statement retrieves all those records whose job_id is neither 670 nor 669.

Ex: SELECT first_name
FROM employee
WHERE job_id NOT IN (670, 669);

BETWEEN clause:

BETWEEN…AND clause is used to check if an expression is between a pair of values. The following statement retrieves all those records whose salary is greater than or equal to 3000 and less than or equal to 5000.

Ex: SELECT first_name
FROM employee
WHERE salary BETWEEN 3000 AND 5000.

NOT BETWEEN:

If NOT is combined with BETWEEN clause, only those records are retrieved which are not in the specified range, including the specified values. The following statement retrieves all the records whose salary is less than 3000 and greater than 5000.

Ex: SELECT first_name
FROM employee
WHERE salary NOT BETWEEN 3000 AND 5000

LOGICAL OPERATORS:

The AND and OR are called Logical operators. They are used to combine multiple conditions in the queries.

'AND' OPERATOR

Returns TRUE only when both conditions are true. The following statement retrieves those records which satisfy both the conditions in the WHERE clause.

Ex: SELECT first_name,last_name FROM employee

WHERE salary = 5000
ANDJob_id IN (670, 669)

'OR' OPERATOR

Returns TRUE when either one of the conditions is true. The following statement retrieves those records which satisfy at least one condition in the WHERE clause.

Ex: SELECT first_name,last_name FROM employee
WHERE salary = 5000
ORJob_id IN (670, 669)

SUBQUERIES

A query (that is, a SELECT statement) may be used as a part of another SQL statement (called the parent, or outer statement), including CREATE TABLE, DELETE, INSERT, SELECT and UPDATE. The results of the child query (also called a sub-query) are passed to the parent SQL statement. A sub-query cannot contain ORDER BY clause.

SINGLE VALUE FROM A SUBQUERY:

The sub-query returns a single value to the outer-query. All of the logical operators that test single values can work with sub-queries, as long as the sub-query returns a single row. The following example illustrates a sub-query, which returns only one value.

```
SELECT first_name, last_name, job_id

FROM employee

WHERE job_id = (SELECT job_id   FROM job

                WHERE function = 'ANALYST')
```

MULTIPLE VALUES FROM A SUB-QUERY:

The sub-query may return one or more rows, the value in the column for each row will be stacked up in a list. The following are some relevant points:

- The sub-query must be enclosed in parentheses.
- Sub-queries that produce more than one row can be used only with many-value (or multiple-row) sub-queries.
- Sub-queries that produce only one row can be used with either single or many value operators.
- BETWEEN cannot be used with a sub-query.

In the following statement, the sub-query returns more than one row to the outer query. So, IN operator is used.

```
SELECT DISTINCT job_id
FROM employee
WHERE employee_id IN (SELECT manager_id FROM employee)
```

NESTED SUBQUERIES:

Nested sub-queries are those, which have a sub-query for an inner query. The following example demonstrates a nested sub-query.

```
SELECT first_name, last_name
 FROM employee
 WHERE salary > (SELECT salary
                 FROM employee
                 WHERE employee_id = (SELECT manager_id
                                      FROM employee
                 WHERE last_name = 'DENNIS')
```

OPERATOR 'ANY':

=ANY is the equivalent of IN. Operator can be any one of '=', '>', '>=', '<', '<=', '!=' and list can be a series of literal strings (such as 'RAVI','KISHORE','RAJU') or a series of literal numbers (such as 2, 43, 76, 32.06, 44) or a column from a sub-query, where each row of the sub-query becomes a member of the list.

```
Ex:        SELECT last_name
             FROM employee
             WHERE department_id=10
                    AND
Job_id =ANY (SELECT  job_id
                                  FROM employee
                               WHERE department_id=20);
```

OPERATOR 'ALL':

!=ALL is the equivalent of NOT IN. Operator can be any of '>', '>=', '<', '<=', '!=' and list can be a series of literal strings (such as 'RAVI','KISHORE','KIRAN'), or a series of literal numbers (such as 2, 23, 76, 32.06, 44), or a column from a sub-query, where each row of the sub-query becomes a member of the list.

```
SELECT last-name
  FROM employee
  WHERE manager-id IN (SELECT employee-id FROM employee
                       WHERE Last-name IN ('Dennis', 'Doyle'))
                   AND Hire-date <ALL (SELECT hire-date FROM employee
                                       WHERE last-name= 'Dennis' or last-name= 'Doyle');
```

CORRELATED SUB-QUERIES:

A correlated sub-query is that which is executed repeatedly once for each value of a candidate row selected by the main query. The outcome of each execution of the sub-query depends on the values of one or more fields in the candidate row; that is, the sub-query is correlated with the main query.

In the following example, the department_id of each row in the main query is correlated with the department_id of the rows in the subquery.

```
SELECT first_name, last_name FROM employee E
WHERE hire_date = (SELECT MIN(hire_date) FROM employee
                   WHERE department_id = E.department_id)
```

EXISTS:

Exists is test for existence. It is placed the way IN might be with a subquery, but differs in two ways. (1) It does not match a column or columns and (2) It is typically used only with a correlated sub-query. EXISTS returns true in a WHERE clause if the sub-query that follows it returns at least one row.

The following example illustrates the efficacy of EXISTS clause.

```
SELECT department_id, name FROM department D
 WHERE EXISTS (SELECT department_id FROM employee
               WHERE department_id = D.department_id)
```

The above query can also be written using IN as follows.

SELECT department_id, name FROM department

WHERE department_id IN (SELECTdepartment_id FROM employee)

NOT EXISTS:

NOT EXISTS is typically used to determine which values in one table do not have matching values in another table. NOT EXISTS allows you to use a correlated a sub-query to eliminate from a table all the records that may successfully be joined to another table

SELECT department_id, name FROM department D

WHERE NOT EXISTS (SELECT department_id FROM employee

WHERE department_id = D.department_id)

GROUP FUNCTIONS:

A GROUP FUNCTION computes a single summary value (such as sum or average) from the individual number values in a group of values. Group functions are useful only in queries and sub-queries.

Group functions ignore NULL's.

AVG( ): gives the average of the values for a group of rows.

SELECT AVG (salary) FROM employee;

COUNT( ): gives the count of rows for a column, or for a table (with *).

SELECT COUNT(*) FROM employee

MIN( ):- gives the minimum of all values for a group for rows.

SELECT MIN (salary) FROM employee;

MAX ( ):- gives the maximum of all values for a group of rows.

SELECT MAX (salary) FROM employee;

SUM( ): gives the sum of all values for a group of rows.

SELECT SUM (salary) FROM employee;

JOINS:

A join combines columns and data from two or more tables (and in rare cases, of one table with itself). The tables are all listed in FROM clause of the SELECT statement, and the relationship between the two tables is specified in the WHERE clause, usually by a simple equality. This is often called an **Equi-Join** because it uses the equal sign in the WHERE clause. The tables can also be joined using other forms of equality, such as >=, < and so on, then it is called a **Non-Equi Join**. If a table is combined with itself then, it is called **Self-Join**.

Equi-Join or Simple Join example:

```
SELECT E.last_name,D.name
FROM employee E, department D
WHERE E.department_id = D.department_id
```

Non-Equi-Join example:

```
SELECT E.first_name, E.last_name, and S.grade_id
FROM employee E, salary_grade S
WHERE E.salary BETWEEN S.lower_bound AND S.upper_bound
```

Self-Join example

```
SELECT M.first_name, M.last_name
FROM employee E, employee M
WHERE E.manager_id = M.employee_id
    AND E.salary>M.salary
```

OUTER JOINS: An OUTER-JOIN is a method for intentionally retrieving selected rows for one table that don't match rows in the other table. (+) must immediately follow the join column of the shorter table, as saying add an extra (NULL) row.

The following example illustrates an outer join.

List the name of departments and employees including which do not have any employee.

SELECT D.department_id, D.name, E.last_name, E.first_name FROM employee E,

department D WHERE E.department_id(+) = D.department_id

GROUP BY:

GROUP BY causes a SELECT to produce one summary row for all selected rows that have identical values in one or more specified columns or expressions. Each expression in the SELECT clause must be one of these things.

- A constant
- A function without parameters (sysdate, user)
- A group function like SUM, AVG, MIN, MAX, COUNT
- A column/expression matching identically to an expression in the GROUP BY clause.

The following example illustrates the GROUP BY clause.

Ex. Find the maximum salary of all employees of their departments?

SELECT department_id, MAX(salary), MIN(salary) FROM employee

GROUP BY department_id

HAVING CLAUSE:

HAVING is used to determine which groups the GROUP BY is to include. A WHERE clause, on the other hand, determines which rows are to be included in groups.

Ex. For each department find the no. of analysts.

SELECT department_id, COUNT(*), AVG(salary) FROM employee

GROUP BY department_idHAVING COUNT(*) > 3

UNION:

UNION combines two SELECT queries. It returns all distinct rows for both SELECT statements. But there are some stipulations.

1. The SELECT's must have the same number of columns.
2. The matching top, bottom columns must be of the same data type (they needn't be of the same length.)

SELECT DISTINCT(department_id) FROM employee

```
GROUP BY department_id

HAVING COUNT(*) > 1

UNION

SELECT department_id

FROM department  d, location l

WHERE d.location_id=l.location_id AND regional_group='KAKINADA';
```

UNION ALL:

When UNION ALL is used, it returns all rows for both SELECT statements, regardless of duplicates.

```
SELECT DISTINCT(department_id) FROM employee

GROUP BY department_id

HAVING COUNT(*) > 1

UNION ALL

SELECT department_id

FROM department  d, location l

WHERE d.location_id=l.location_id AND regional_group='RAJAMUNDRY';
```

INTERSECT:

INTERSECT combines two queries and returns only those rows from the first SELECT statement that are identical to at least one row from the second SELECT statement. The number of columns and data types must be identical between SELECT statements.

```
SELECT department_id

FROM employee

GROUP BY department_id

HAVING COUNT(*) > 1

INTERSECT

SELECT department_id
```

FROM department d, location l

WHERE d.location_id=l.location_id AND regional_group = ‘VISAKHAPATNAM’;

MINUS:

MINUS combines two queries. It returns only those rows from the first SELECT statement that are not produced by the second SELECT statement (the first SELECT MINUS the second SELECT). The number of columns and data types must be identical between SELECT statements, although the names of

the columns need not be same.

SELECT department_id

FROM employee

GROUP BY department_id

HAVING COUNT(*) > 1

MINUS

SELECT department_id

FROM department d,location l

WHERE d.location_id=l.location_id AND regional_group = ‘CHICAGO’ ;

ORDER BY *POSITION:*

The resultant rows from SELECT statements can also be sorted in ascending or descending order by specifying the position of the column to be sorted.

(SELECT department_id,job_id

FROM employee

WHERE department_id=20

UNION

SELECT department_id,job_id

FROM employee

WHERE department_id=10)ORDER BY 2;