Fundamental Queries

Select Statement

Select command is used to view the records from the table. To view all the columns and all the rows '*'can be specified with the select statement. The name of the table is required while specifying the select.

```
Syntax :-  Select * from <tablename>;
```

The introductory SELECT statement has three clauses: SELECT FROM WHERE

The SELECT clause specifies the table columns that are retrieved. (Projection in relational algebra)
The FROM clause specifies the tables accessed (cartesian product in relational algebra)
The WHERE clause filters the rows from the table. The WHERE clause is optional; if missing, all table rows are used. (selection in relational algebra)

For Example:

```
SELECT column_name,column_name FROM table_name;
SELECT * FROM table_name;
```

Where Statement

The WHERE clause is used to extract only those records that fulfill a specified criterion. It can also be used with UPDATE and DELETE commands.

Syntax:

```
SELECT column_name
FROM  table_name
WHERE condition ;
```

AND,OR and NOT Statement

AND is an operator that joins two conditions. Both conditions must be true for the row to be included in the result set.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_1 = value_1
 AND column_2 = value_2;
```

The **OR** operator displays a record if either the first condition OR the second condition is true.

Syntax:

```
 SELECT column_name
 FROM table_name
 WHERE column_name = value_1
   OR column_name = value_2;
```

The **NOT** operator can put before any conditional statement to select rows for which that statement is false. NOT is commonly used with LIKE.

Syntax:

```
SELECT column_name
FROM table_name
WHERE NOT condition;
```

Handling NULL

The NULL keyword can also be used in predicates to check for null values.NULL value field is a valueless field.
If the field in the table is not selected, it is possible to add a new record or update the record without adding value to this field. After that, the field will be saved at NULL value.

Null value is identified with **Is NULL** OR **NOT NULL** Keyword.
IS NULL and IS NOT NULL are operators used with the WHERE clause to test for empty values.

Syntax :

```
SELECT column_name(s)
FROM table_name
WHERE column_name IS NULL(NOT NULL);
```

IN, BETWEEN and LIKE

BETWEEN returns values that come within a given range. BETWEEN operator is inclusive: begin and end values are included.

Syntax of BETWEEN:

```
SELECT column_name
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

The **IN** keyword allows you to easily test if the expression matches any value in the list. It is used to help reduce the need for multiple OR cases.

Syntax for IN:

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

The LIKE operator is used to search for a specified pattern in a column.
There are two operators that are used with the LIKE operator.

1. %: Percentage is used for representation of single, multiple or no occurrence.
2. _: The underscore is used for representation of a single character.

Syntax for LIKE:
SELECT column_name
FROM table_name
WHERE column_name ***LIKE*** pattern;

ORDER BY Statement

ORDER BY is a statement that sorts the result set by a particular column either ascending or descending.
It sorts the result in ascending order by default. For descending order, use DESC KEYWORD.

Syntax:
SELECT column_name
FROM table_name
ORDER BY column_name ASC | DESC;

LIMIT and OFFSET

LIMIT is a keyword that lets you specify the maximum number of rows the result set will have.

Syntax for LIMIT:

SELECT column_name(s)
FROM table_name
LIMIT number;

The OFFSET keyword is used to identify the starting point to retrieve lines from the result set. Basically, it releases the first set of records.

It is only used with ORDER BY.
Value must be greater than or equal to zero. It cannot be negative, else return error.

Syntax For OFFSET:

```
SELECT column_name
FROM table_name
WHERE condition
ORDER BY column_name
OFFSET rows_name ROWS;
```

DISTINCT

DISTINCT specifies that the statement is going to be a query that returns unique values in the specified column(s).

Syntax:

```
SELECT DISTINCT column_name
FROM table_name;
```

CASE EXPRESSION

The CASE statement returns value when the original condition is met Therefore, if the situation is true, we will stop reading and return the results. If there are no valid terms, it returns the value in the ELSE clause.

If it is not part of ELSE and there are no true terms, it returns NULL.

Syntax:

```
SELECT column_name,
 CASE
```

```
   WHEN condition THEN 'Result_1'
   WHEN condition THEN 'Result_2'
   ELSE 'Result_3'
  END
FROM table_name;
```

GROUP BY

The **GROUP BY** clause is used to group rows that have the same column values.

Syntax:

```
SELECT column_name
FROM table_name
WHERE condition
GROUP BY column_name;
```

HAVING CLAUSE

The **HAVING clause** is similar to where condition, but the WHERE keyword could not be used with aggregate functions.To group data with aggregate function, HAVING is used.

Syntax:

```
SELECT column_name, COUNT(*)
FROM table_name
GROUP BY column_name
HAVING COUNT(*) > value;
```

Subquery

A subquery is a SQL query nested inside a larger query.

Where does the Subquery occur?
A subquery may occur in:

- A SELECT clause
- A WHERE clause

The subquery can be nested inside a SELECT,INSERT, UPDATE, or DELETE statement or inside another subquery.
A subquery is usually written inside the WHERE clause of another SQL SELECT statement.
You can use the comparison operators, such as >, <, or =. The comparison operator can also be a multiple-row operator, such as IN, ANY, or ALL.

Syntax:
```
SELECT column_name [, column_name ]
FROM   table1 [, table2 ]
WHERE  column_name OPERATOR
   (SELECT column_name [, column_name ]
   FROM table1 [, table2 ]
   [WHERE])
```

Example:
Display the price and average price of all books:
SELECT Price, **(SELECT Avg(Price) FROM titles)** AS AveragePrice FROM titles

Types of Subquery

- Non Correlated Subqueries
- Correlated Subqueries

Non Correlated Subqueries

A noncorrelated subquery executes independently of the outer query. The subquery executes first, and then passes its results to the outer query.
For example:
SELECT name, street, city, state FROM addresses WHERE state IN (SELECT state FROM states);

A query's WHERE and HAVING clauses can specify non correlated subqueries if the subquery resolves to a single row, as shown below:

In WHERE clause

```
SELECT COUNT(*) FROM SubQ1 WHERE SubQ1.a = (SELECT y from SubQ2);
```

In HAVING clause

```
SELECT COUNT(*) FROM SubQ1 GROUP BY SubQ1.a HAVING SubQ1.a = (SubQ1.a &
(SELECT y from SubQ2)
```

Correlated Subqueries

A correlated subquery typically obtains values from its outer query before it executes. When the subquery returns, it passes its results to the outer query.
A correlated subquery refers to one or more columns from outside of the subquery.

In the following example, the subquery needs values from the addresses.state column in the outer query:

```
SELECT name, street, city, state FROM addresses
    WHERE EXISTS (SELECT * FROM states WHERE states.state = addresses.state);
```