Introduction Material:

View

A view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database

Syntax:

CREATE VIEW *view_name* AS
SELECT *column1*, *column2*, ...
FROM *table_name*
WHERE *condition*;

GROUP BY

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);*

HAVING

The HAVING clause was added to SQL because the WHERE keyword could not 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);*

ORDER BY

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

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

WITH Clause

The SQL WITH clause allows you to give a sub-query block a name (a process also called sub-query refactoring), which can be referenced in several places within the main SQL query.

WITH temporaryTable (averageValue) as

SELECT avg(Attr1)

FROM Table),

SELECT Attr1

FROM Table

WHERE Table.Attr1 > temporaryTable.averageValue

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.

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

Recursive Query
A recursive common table expression (CTE) is a CTE that references itself. By doing so, the CTE repeatedly executes, returns subsets of data, until it returns the complete result set.

```
WITH expression_name (column_list)
AS
(
   -- Anchor member
   initial_query
   UNION ALL
   -- Recursive member that references expression_name.
   recursive_query
)
-- references expression name
SELECT *
FROM   expression_name
```

1. Views

    1. Create a view of all customers from Country “Brazil”.

        Customer (ID, NAME, CITY, COUNTRY)

    2. Create a view that selects every product in the "Products" table with a price higher than the average price

        Products (ID, NAME , PRICE)

2. Group by - having -Order by statements

    1. List the number of customers in each country. Only include countries with more than 5 customers

        Customer (ID, NAME, CITY, COUNTRY)

    2. List the number of customers in each country, sorted high to low (Only include countries with more than 5 customers)

        Customer (ID, NAME, CITY ,COUNTRY)

3. WITH clause

1. Find all the employee whose salary is more than the average salary of all employees.

Employee(EMPLOYEEID, NAME, SALARY)

2. Find all the airlines where the total salary of all pilots in that airline is more than the average of total salary of all pilots in the database.

Pilot (EMPLOYEEID , AIRLINE , NAME , SALARY)

4. SQL CASE statement

1. Display employee names and grade based on their salary as ,“Grade 1” if salary >5000 ; “Grade 2” if 200< salary <500 ; “Grade 3” if salary >200

Employee(EMPLOYEEID,NAME,SALARY)

2. Order customers by City. However, if City is NULL, then order by Country

Customer(ID,NAME,CITY,COUNTRY)

5. Recursive query

1. Recursively query all sub-categories for parent “Database software”
2. Recursive query which generate a series of first 5 odd numbers

id	name	parent_category
1	Root Node	(null)
2	Software	1
3	Hardware	1
4	Notebooks	3
5	Phones	3
6	Applications	2
7	Database Software	2
8	Relational DBMS	7
9	Tools	7
10	Commandline tools	9
11	GUI Tools	9
12	Android Phones	5
13	iPhone	5
14	Windows Phones	5

1.1

```
CREATE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName
FROM Customers
WHERE Country = "Brazil";

SELECT * FROM [Brazil Customers];
```

1.2

```
CREATE VIEW [Products Above Average Price] AS
SELECT ProductName, Price
FROM Products
WHERE Price > (SELECT AVG(Price) FROM Products);

SELECT * FROM [Products Above Average Price];
```

2.1

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

2.2

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

3.1

```
WITH temporaryTable(averageValue) as
    (SELECT avg(Salary)
    from Employee),
        SELECT EmployeeID,Name, Salary
        FROM Employee, temporaryTable
         WHERE Employee.Salary >

 temporaryTable.averageValue;
```

3.2

```
WITH totalSalary(Airline, total) as
    (SELECT Airline, sum(Salary)
    FROM Pilot
    GROUP BY Airline),
    airlineAverage(avgSalary) as
    (SELECT avg(Salary)
    FROM Pilot )
    SELECT Airline
    FROM totalSalary, airlineAverage
     WHERE totalSalary.total >

 airlineAverage.avgSalary;
```

4.1

```
SELECT name, CASE WHEN salary < 200 THEN 'GRADE 1'
                  WHEN salary > 200 AND salary < 5000 THEN 'GRADE 2'
                  ELSE 'GRADE 3'
             END CASE
FROM employee;
```

4.2

```
SELECT Name, City, Country
FROM Customer
ORDER BY
(CASE
    WHEN City IS NULL THEN Country
    ELSE City
END);
```

5.1

```
with recursive cat_tree as (
    select id,
           name,
           parent_category
    from category
    where name = 'Database Software'  -- this defines the start of the recursion
    union all
    select child.id,
           child.name,
           child.parent_category
    from category as child
      join cat_tree as parent on parent.id = child.parent_category -- the self join to the CTE builds up the recursion
)
select *
 from cat_tree;
```

id	name	parent_category
7	Database Software	2
8	Relational DBMS	7
9	Tools	7
10	Commandline tools	9
11	GUI Tools	9

5.2

```
WITH RECURSIVE
 odd_no (sr_no, n) AS
 (
 SELECT 1, 1
 union all
 SELECT sr_no+1, n+2 from cte where odd_no < 5
 )
   SELECT * FROM odd_no;
```