Database Management System – cs422 DE

Assignment 3 – Week 3 & 4

This assignment is based on lecture 3 & 4 (chapter 6 & 7).

- Submit your *own work* on time. No credit will be given if the assignment is submitted after the due date.
- Note that the completed assignment should be submitted in .doc, .docx, .rtf or .pdf format only.
- In MCQs, if you think that your answer needs explanation to get credit then please write it down.
- You are encouraged to discuss these questions in the Sakai forum.

---

1) The database schema is written in

(A) HLL (B) DML (C) DDL (D) DCL

ANS: C

2) The language used in application programs to request data from the DBMS is referred to as

(A) DML (B) DDL (C) VDL (D) SDL

ANS: A

3) Count function in SQL returns the number of

(A) values (B) distinct values (C) groups (D) columns

ANS: A

4) 'AS' clause is used in SQL for

(A) Selection (B) Rename (C) Join (D) Projection

ANS: B

5) Which is not a DDL statement ?

(A) Create (B) Alter (C) Delete (D) Drop

ANS: C

6) The statement in SQL which allows to change the definition of a table is

(A) Alter (B) Update (C) Create (D) Select

ANS: A

7) What restrictions apply to the use of the aggregate functions within the SELECT statement? How do nulls affect the aggregate functions?

ANS: Aggregate functions in SQL, such as COUNT, SUM, AVG, MIN, and MAX, are used to perform operations on sets of values. Here are some general restrictions and considerations for using aggregate functions in the SELECT statement:

i) Grouping Columns: When using aggregate functions, it is common to include a GROUP BY clause in the SELECT statement. Columns in the SELECT list that are not part of the aggregate functions must be included in the GROUP BY clause.

Example:

```
SELECT department, AVG(salary)
FROM employees
GROUP BY department;
```

Ii) Filtering with HAVING: If you want to filter the results of aggregate functions based on a condition, you use the HAVING clause. The HAVING clause is similar to the WHERE clause but is specifically used with aggregate functions.

Example:

```
SELECT department, AVG(salary)
FROM employees
GROUP BY department
HAVING AVG(salary) > 50000;
```

ii) Handling NULL Values: Aggregate functions generally ignore NULL values, except for the COUNT(*) function, which counts all rows regardless of NULL values.

Example:

```
SELECT AVG(column_name)
FROM table_name;
```


In the above example, NULL values in the specified column will be ignored when calculating the average.

If you want to include NULL values in the calculations, you can use the COALESCE or IFNULL function to replace NULL with a default value.

Example:

```
SELECT AVG(COALESCE(column_name, 0))
FROM table_name;
```

8) List the order in which the WHERE, GROUP BY, and HAVING clauses are executed by the database in the following SQL statement.

```
SELECT section_id, COUNT(*), final_grade
FROM enrollment
WHERE TRUNC(enroll_date) > TO_DATE('2/16/2003', 'MM/DD/YYYY')
GROUP BY section_id, final_grade HAVING COUNT(*) > 5
```

ANS:

i) FROM
ii) WHERE TRUNC
iii) GROUP BY
iv) HAVING COUNT
v) SELECT
vi) COUNT

9) Explain how the GROUP BY clause works. What is the difference between WHERE and HAVING clauses?

ANS: The GROUP BY clause in SQL is used to group rows that have the same values in specified columns into summary rows, like in the case of aggregate functions. It is often used in conjunction with aggregate functions such as COUNT, SUM, AVG, MAX, or MIN to perform operations on each group of rows.

ii) Grouping Columns: You specify one or more columns in the GROUP BY clause based on which you want to group the rows.

Example:

```
SELECT department, AVG(salary)
FROM employees
GROUP BY department;
```

In this example, the rows are grouped by the "department" column.

Aggregate Functions: You can use aggregate functions in the SELECT statement to perform calculations on each group of rows.

Example:

```
SELECT department, AVG(salary)
FROM employees
GROUP BY department;
```

Here, the AVG(salary) calculates the average salary for each department.

ii) The HAVING clause is used in combination with the GROUP BY clause to filter the groups based on specified conditions. It is similar to the WHERE clause, but the WHERE clause is used to filter individual rows before they are grouped, while the HAVING clause is used to filter groups after they have been formed by the GROUP BY clause.

Here's a comparison between WHERE and HAVING:

WHERE Clause:

Filters individual rows before they are grouped.
Used with non-aggregated columns.
Typically used for row-level conditions.

Example:

```
SELECT department, AVG(salary)
FROM employees
WHERE salary > 50000
GROUP BY department;
HAVING Clause:
```

Filters groups after they have been formed by the GROUP BY clause.
Used with aggregated columns or aggregate functions.
Typically used for group-level conditions.
Example:

```
SELECT department, AVG(salary)
FROM employees
GROUP BY department
```

HAVING AVG(salary) > 50000;

In summary, the GROUP BY clause is used to group rows based on specified columns, and the HAVING clause is used to filter the groups based on conditions involving aggregate functions. The WHERE clause, on the other hand, filters individual rows before any grouping occurs.

Can the ANY and ALL operators be used on the DATE data type? Write a simple query to prove your answer.

ANS: The ANY and ALL operators in SQL are typically used in conjunction with subqueries to compare a value to a set of values. However, they are not commonly used directly with the DATE data type. Instead, they are more commonly used with numerical or string comparisons.

Here's an example using the ANY operator with a subquery involving the DATE data type:

SELECT employee_id, hire_date

FROM employees

WHERE hire_date > ANY (SELECT hire_date FROM employees WHERE department_id = 30);

In this example, the query retrieves employee IDs and hire dates from the employees table for employees whose hire date is greater than any hire date in the subquery for employees in department 30.

Similarly, you can use the ALL operator in a similar manner:

SELECT employee_id, hire_date

FROM employees

WHERE hire_date > ALL (SELECT hire_date FROM employees WHERE department_id = 30);

This query retrieves employee IDs and hire dates for employees whose hire date is greater than all hire dates in the subquery for employees in department 30.

While these examples involve the DATE data type indirectly through the hire_date column, the ANY and ALL operators themselves are not inherently tied to the DATE data type. They can be used with various data types, depending on the context of the comparison.

10) The following SQL lists staffs who work in branch at '163 Main St'.

```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE branchNo =
      (SELECT branchNo
      FROM Branch
      WHERE street = '163 Main St');
```

Will there be any problem with this query if there is more than one branch at '163 Main St'? If yes, then explain the problem and right down the correct query.

ANS: Yes, there could be a problem with the given query if there is more than one branch at '163

Main St'. The subquery in the WHERE clause is expected to return a single value, and if it returns multiple values (due to multiple branches at '163 Main St'), it will result in an error.

To handle the scenario where there could be multiple branches at '163 Main St', you can use the IN or EXISTS clause in the WHERE clause to check if the branchNo from the Staff table is in the list of branchNos for the specified street. Here's how you can modify the query:

```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE branchNo IN
  (SELECT branchNo
   FROM Branch
   WHERE street = '163 Main St');
```

or using EXISTS:

```
SELECT staffNo, fName, lName, position
FROM Staff s
WHERE EXISTS
  (SELECT 1
   FROM Branch b
   WHERE b.branchNo = s.branchNo
    AND b.street = '163 Main St');
```


Both of these queries handle the case where there are multiple branches at '163 Main St' by using the IN or EXISTS clause to check if the branchNo from the Staff table matches any of the branchNo for the specified street.

What is Referential integrity constraint?

ANS: Referential integrity is a concept in relational databases that ensures the consistency and accuracy of data by maintaining the relationships between tables. A referential integrity constraint is a rule that is applied to relationships between tables to ensure that they remain valid. The primary purpose of referential integrity constraints is to maintain the integrity and accuracy of data within the database.

There are two main components of referential integrity:

i)Primary Key-Foreign Key Relationship:

In a relational database, a primary key uniquely identifies each record in a table.

A foreign key is a field in a table that refers to the primary key in another table. It establishes a link between the two tables.

ii)Referential Integrity Constraints:

Foreign Key Constraint: This is the most common type of referential integrity constraint. It ensures that values in the foreign key column of a child table match the values in the primary key column of the parent table.

CASCADE: If a record in the parent table is deleted or updated, the corresponding records in the child table can be automatically deleted or updated to maintain consistency.

SET NULL or SET DEFAULT: Instead of deleting or updating related records, the foreign key values in the child table can be set to NULL or a default value.

NO ACTION or RESTRICT: This prevents the deletion or update of a record in the parent table if there are related records in the child table.

Example:

Consider two tables, "Customers" and "Orders." The "Customers" table has a primary key "CustomerID," and the "Orders" table has a foreign key "CustomerID" that references the primary key in the "Customers" table.

```
CREATE TABLE Customers (
  CustomerID INT PRIMARY KEY,
  CustomerName VARCHAR(255)
);
 CREATE TABLE Orders (
  OrderID INT PRIMARY KEY,
  OrderDate DATE,
  CustomerID INT,
  FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
  ON DELETE CASCADE
  ON UPDATE NO ACTION
);
```

What is the difference between primary key and unique key?
ANS: Both primary keys and unique keys are used to enforce the uniqueness of values within a column or a set of columns in a relational database, but they serve slightly different purposes:

Primary Key:

A primary key is a column or set of columns in a table that uniquely identifies each record in that table.

It must contain unique values, and it cannot contain NULL values.

There can be only one primary key in a table.

A primary key is used to establish relationships between tables, serving as a foreign key in related tables.

The primary key constraint automatically creates a unique index on the specified column(s).

Example:

```
CREATE TABLE Employees (
  EmployeeID INT PRIMARY KEY,
  FirstName VARCHAR(50),
  LastName VARCHAR(50)
);
```


Unique Key:

A unique key is a constraint that ensures the values in a column or a set of columns are unique across all the records in the table.

Unlike a primary key, a unique key can allow NULL values (depending on the database system and configuration).

A table can have multiple unique keys.

While unique keys enforce uniqueness, they do not automatically create a relationship between tables.

Example:

```
CREATE TABLE Students (
  StudentID INT,
  StudentName VARCHAR(50),
  Email VARCHAR(100) UNIQUE
```

```
);
```

11) **Solve the question 7.10 from the course text book (5th edition).**

ANS:

```
CREATE TABLE hotel_tbl (

 id INT PRIMARY_KEY,

name VARCHAR(120),

address VARCHAR(250),

rating VARCHAR(20)

) ;
```

12) **Solve the question 7.12 from the course text book (5th edition).**

ANS:

```
CREATE TABLE booking_history_tbl (
bookingId INT PRIMARY KEY,
roomId INT,
guestId INT,
price DECIMAL(6,2),
type VARCHAR(20),
dateFrom DATE,
dateTo DATE,
FOREIGN KEY(roomId) REFERENCES room_tbl(roomId),
FOREIGN KEY(guestId) REFERENCES guest_tbl(guestId)
);

INSERT INTO booking_history_tbl
SELECT * FROM booking_tbl
WHERE dateTo < '01/01/2007';

DELETE FROM booking_tbl
WHERE dateTo < '01/01/2007';
```