Lecture 7
More SQL

Agenda

- Aggregate Functions
- Grouping/Having
- Order by/Substring
  Comparison/Arithmetic operations
- Modifications in SQL
- Specifying Constraints as Assertions
- Triggers in SQL
- Views in SQL

Aggregate Functions

AGGREGATE FUNCTIONS

- Include **COUNT, SUM, MAX, MIN, and AVG**
- **Query 1:** Find the maximum salary, the minimum salary, and the average salary among all employees.

```
Q1:   SELECT  MAX(SALARY), MIN(SALARY),
              AVG(SALARY)
      FROM    EMPLOYEE
```

- Some SQL implementations *may not allow more than one function* in the SELECT-clause

AGGREGATE FUNCTIONS

- Query 2: Find the maximum salary, the minimum salary, and the average salary among employees who work for the 'Research' department.

```
Q2:   SELECT  MAX(SALARY), MIN(SALARY),
              AVG(SALARY)
      FROM    EMPLOYEE, DEPARTMENT
      WHERE   DNO=DNUMBER AND
              DNAME='Research'
```

AGGREGATE FUNCTIONS

- **Query 3:** Retrieve the total number of employees in the company.

```
Q3:   SELECT  COUNT (*)
      FROM    EMPLOYEE
```

- **Query 4:** Retrieve the number of employees in the 'Research' department

```
Q4:   SELECT  COUNT (*)
      FROM    EMPLOYEE, DEPARTMENT
      WHERE   DNO=DNUMBER AND
              DNAME='Research'
```

Grouping/Having

GROUPING

- In many cases, we want to apply the aggregate functions to *subgroups of tuples* in a relation
- Each subgroup of tuples consists of the set of tuples that have the *same value* for the *grouping attribute(s)*
- The function is applied to each subgroup independently
- SQL has a **GROUP BY**-clause for specifying the grouping attributes, which *must also appear in the SELECT-clause*

GROUPING

- **Query 5: For each department**, retrieve the department number, the number of employees in the department, and their average salary.

```
Q5:   SELECT   DNO, COUNT (*), AVG (SALARY)
      FROM     EMPLOYEE
      GROUP BY DNO
```

- In Q5, the EMPLOYEE tuples are divided into groups-
  - Each group having the same value for the grouping attribute DNO
- The COUNT and AVG functions are applied to each such group of tuples separately
- The SELECT-clause includes only the grouping attribute and the functions to be applied on each group of tuples
- A join condition can be used in conjunction with grouping

GROUPING

- **Query 6:** For each project, retrieve the project number, project name, and the number of employees who work on that project.

```
Q6:   SELECT    PNUMBER, PNAME, COUNT (*)
      FROM      PROJECT, WORKS_ON
      WHERE     PNUMBER=PNO
      GROUP BY  PNUMBER, PNAME
```

- In this case, the grouping and functions are applied after the joining of the two relations

THE HAVING-CLAUSE

- Sometimes we want to retrieve the values of these functions for only those *groups that satisfy certain conditions*

- The **HAVING**-clause is used for specifying a selection condition on groups (rather than on individual tuples)

THE HAVING-CLAUSE

- **Query 7:** For each project *on which more than two employees work*, retrieve the project number, project name, and the number of employees who work on that project.

```
Q7:   SELECT    PNUMBER, PNAME, COUNT(*)
      FROM      PROJECT, WORKS_ON
      WHERE     PNUMBER=PNO
      GROUP BY  PNUMBER, PNAME
      HAVING    COUNT (*) > 2
```

Order by/ Substring Comparison/ Arithmetic operations

ORDER BY

- The **ORDER BY** clause is used to sort the tuples in a query result based on the values of some attribute(s)
- **Query 8:** Retrieve a list of employees and the projects each works in, ordered by the employee's department, and within each department ordered alphabetically by employee last name.

```
Q8:   SELECT    DNAME, LNAME, FNAME, PNAME
      FROM      DEPARTMENT, EMPLOYEE,
                WORKS_ON, PROJECT
      WHERE     DNUMBER=DNO AND SSN=ESSN
                AND PNO=PNUMBER
      ORDER BY  DNAME, LNAME
```

ORDER BY

- The default order is in ascending order of values
- We can specify the keyword **DESC** if we want a descending order; the keyword **ASC** can be used to explicitly specify ascending order, even though it is the default
- Example: Same Q8, but sort in descending order

```
Q9:   SELECT    DNAME, LNAME, FNAME, PNAME
      FROM      DEPARTMENT, EMPLOYEE,
                WORKS_ON, PROJECT
      WHERE     DNUMBER=DNO AND SSN=ESSN
                AND PNO=PNUMBER
      ORDER BY  DNAME, LNAME  DESC
```

SUBSTRING COMPARISON

- The **LIKE** comparison operator is used to compare partial strings

- Two reserved characters are used:
  - **'%'** (or **'*'** in some implementations)
    - replaces an arbitrary number of characters,
  - **'_'**
    - replaces a single arbitrary character

SUBSTRING COMPARISON

- **Query 10:** Retrieve all employees whose address is in Houston, Texas.
- Here, the value of the ADDRESS attribute must contain the substring 'Houston,TX' in it.

```
Q10: SELECT   FNAME, LNAME
     FROM     EMPLOYEE
     WHERE    ADDRESS LIKE '%Houston,TX%'
```

SUBSTRING COMPARISON

- **Query 11:** Retrieve all employees who were born during the 1950s.
  - Here, '5' must be the 8th character of the string (according to our format for date), so the BDATE value is '_____5_', with each underscore as a place holder for a single arbitrary character.

```
Q11:    SELECT   FNAME, LNAME
        FROM     EMPLOYEE
        WHERE    BDATE LIKE   '_____5_'
```

- The LIKE operator allows us to get around the fact that each value is considered atomic and indivisible
  - Hence, in SQL, character string attribute values are not atomic

ARITHMETIC OPERATIONS

- The standard arithmetic operators **'+', '-', '*', and '/'** (for addition, subtraction, multiplication, and division, respectively) can be applied to numeric values in an SQL query result
- **Query 12: Show the effect** of giving all employees who work on the 'ProductX' project a 10% raise.

```
Q12:    SELECT   FNAME, LNAME, 1.1*SALARY
        FROM     EMPLOYEE, WORKS_ON, PROJECT
        WHERE    SSN=ESSN AND PNO=PNUMBER
                 AND PNAME='ProductX'
```

Summary of SELECT SQL Queries

- A SELECT query in SQL can consist of up to six clauses, but only the first two, SELECT and FROM, are mandatory. The clauses are specified in the following order:

```
SELECT      <attribute list>
FROM        <table list>
[WHERE      <condition>]
[GROUP BY   <grouping attribute(s)>]
[HAVING     <group condition>]
[ORDER BY   <attribute list>]
```

Modifications in SQL

Modifications in SQL

- There are three SQL commands to modify the database:
  - **INSERT**
  - **DELETE**
  - **UPDATE**

INSERT

- In its simplest form, it is used to add one or more tuples to a relation

- Attribute values should be listed in the *same order* as the attributes were specified in the **CREATE TABLE** command

INSERT

- Example:

I1: **INSERT INTO** EMPLOYEE
VALUES ('Richard','K','Marini', '653298653', '30-DEC-52','98 Oak Forest,Katy,TX', 'M', 37000,'987654321', 4)

- An alternate form of INSERT specifies explicitly the attribute names that correspond to the values in the new tuple
  - Attributes with NULL values can be left out
- Example: Insert a tuple for a new EMPLOYEE for whom we only know the FNAME, LNAME, and SSN attributes.

I2: **INSERT INTO** EMPLOYEE (FNAME, LNAME, SSN)
VALUES ('Richard', 'Marini', '653298653')

DELETE

- Removes tuples from a relation:
  - Includes a WHERE-clause to select the tuples to be deleted
  - Referential integrity should be enforced
  - Tuples are deleted from only *one table* at a time (unless CASCADE is specified on a referential integrity constraint)
  - A missing WHERE-clause specifies that *all tuples* in the relation are to be deleted; the table then becomes an empty table
  - The number of tuples deleted depends on the number of tuples in the relation that satisfy the WHERE-clause

DELETE

- Examples:

D1:
```
DELETE FROM  EMPLOYEE
WHERE        LNAME='Brown'
```

D2:
```
DELETE FROM  EMPLOYEE
WHERE        SSN='123456789'
```

D3:
```
DELETE FROM  EMPLOYEE
WHERE        DNO  IN
             ( SELECT   DNUMBER
               FROM     DEPARTMENT
               WHERE    DNAME='Research')
```

D4:
```
DELETE FROM  EMPLOYEE
```

UPDATE

- Used to modify attribute values of one or more selected tuples
- A WHERE-clause selects the tuples to be modified
- An additional SET-clause specifies the attributes to be modified and their new values
- Each command modifies tuples *in the same relation*
- Referential integrity should be enforced

UPDATE

- **Example**: Change the location and controlling department number of project number 10 to 'Bellaire' and 5, respectively.

```
U1:     UPDATE PROJECT
        SET    PLOCATION = 'Bellaire', DNUM = 5
        WHERE  PNUMBER=10
```

UPDATE

- **Example: Give** all employees in the 'Research' department a 10% raise in salary.

```
U2:     UPDATE   EMPLOYEE
        SET      SALARY = SALARY *1.1
        WHERE    DNO  IN    (SELECT   DNUMBER
                             FROM      DEPARTMENT
                             WHERE     DNAME='Research')
```

- In this request, the modified SALARY value depends on the original SALARY value in each tuple
  - The reference to the SALARY attribute on the right of = refers to the old SALARY value before modification
  - The reference to the SALARY attribute on the left of = refers to the new SALARY value after modification

Specifying Constraints as Assertions

Specifying Constraints as Assertions

- In this section, we introduce an additional feature of SQL: the **CREATE ASSERTION** statement.

- CREATE ASSERTION, which can be used to specify additional types of constraints that are outside the scope of the *built-in relational model constraints* (primary and unique keys, entity integrity, and referential integrity).

Specifying Constraints as Assertions

- In SQL, users can specify general constraints using the **CREATE ASSERTION** statement.

- Each assertion is given a constraint name and is specified via a condition similar to the WHERE clause of an SQL query.

Specifying Constraints as Assertions

- Example: Specify the constraint that *the salary of an employee must not be greater than the salary of the manager of the department that the employee works for* in SQL.
- **CREATE ASSERTION** SALARY_CONSTRAINT

  CHECK

  (**NOT EXISTS** (**SELECT** *

  FROM EMPLOYEE E, EMPLOYEE M,

  DEPARTMENT D

  WHERE E.Salary>M.Salary

  AND E.Dno = D.Dnumber

  AND D.Mgr_ssn = M.Ssn));

Specifying Constraints as Assertions

- The constraint name SALARY_CONSTRAINT is followed by the keyword CHECK, which is followed by a **condition** in parentheses that must hold true on every database state for the assertion to be satisfied.
- The constraint name can be used later to disable the constraint or to modify or drop it.
- The **DBMS** is responsible for ensuring that the condition is not violated.

Triggers in SQL

Triggers in SQL

- In many cases it is convenient to specify the type of action to be taken when certain events occur and when certain conditions are satisfied.
- For example, it may be useful to **specify a condition** that, if violated, causes some user to be informed of the violation.
- The action that the **DBMS** must take in this case is to send an appropriate message to that user. The condition is thus used to **monitor** the database.
- This is done by CREATE TRIGGER statement in SQL.

Triggers in SQL

- Suppose we want to check whenever an employee's salary is greater than the salary of his or her direct supervisor in the COMPANY database.
- Several events can trigger this rule: inserting a new employee record, changing an employee's salary, or changing an employee's supervisor which will notify the supervisor.

Triggers in SQL

- **CREATE TRIGGER** SALARY_VIOLATION

BEFORE INSERT OR UPDATE OF
SALARY, SUPERVISOR_SSN **ON** EMPLOYEE

FOR EACH ROW
WHEN (**NEW**.SALARY > (**SELECT** SALARY
FROM EMPLOYEE
WHERE SSN=
NEW.SUPERVISOR_SSN))

INFORM_SUPERVISOR(**NEW**.Supervisor_ssn, **NEW**.Ssn);

Triggers in SQL

- A typical trigger which is regarded as an ECA (Event, Condition, Action) rule has three components:

1. The **Event**: These are usually database modify operations. BEFORE or AFTER can be used to specify when the trigger is executed (i.e. before or after the operation).
2. The **Condition:** Determines whether the action should be executed. It is specified in the WHEN clause of the trigger. If *no condition* is specified, the action will be executed once the event occurs.
3. The **Action:** The action is usually a sequence of SQL statements, but it could also be an external program that will be automatically executed. In this example, the action is to execute the stored procedure INFORM_SUPERVISOR.

Views in SQL

Views in SQL

- A **view** in SQL terminology is a single table that is derived from other tables. These other tables can be *base tables* or previously defined views.
- A view does not necessarily exist in physical form; it is considered to be a **virtual table.**
- This limits the possible update operations that can be applied to views

Views in SQL

- We can think of a view as a way of specifying a table that we need to reference frequently, even though it may not exist physically.
- For example, referring to the COMPANY database, we may frequently issue queries that retrieve the employee name and the project names that the employee works on. Rather than having to specify the join of the three tables EMPLOYEE, WORKS_ON, and PROJECT every time we issue this query, we can define a view that is specified as the result of these joins. Then we can issue queries on the view.

Views in SQL

- In SQL, the command to specify a view is **CREATE VIEW**. The view is given a (virtual) table *name* (or view name), a *list of attribute names*, and a *query* to specify the contents of the view.
- Example:

```
CREATE VIEW WORKS_ON1 (FN, LN, PN, H)
AS SELECT Fname, Lname, Pname, Hours
   FROM EMPLOYEE, PROJECT, WORKS_ON
   WHERE Ssn = Essn AND Pno = Pnumber;
```

Views in SQL

- We can now specify SQL queries on a view:

```
SELECT Fname, Lname
FROM   WORKS_ON1
WHERE  Pname = 'ProductX';
```

- If we do not need a view anymore, we can use the **DROP VIEW:**

```
DROP VIEW WORKS_ON1;
```

View in SQL

- Updating base tables: Different strategies as to when a view is updated:
  - Immediate Update
  - Lazy Update
  - Periodic Update
- Updating Views:
  - Issuing an INSERT, DELETE, or UPDATE command on a view table is in many cases not possible.

Thank You