SQL: Data Update & View Definition

CS 377: Database Systems

Recap: SQL Queries

```
SELECT      [DISTINCT] <attribute list>
FROM         <table list>
[WHERE       <condition on the tables>]
[GROUP BY <grouping attributes>]
[HAVING      <group condition>]
[ORDER BY  <attribute list>   ASC | DESC]
[LIMIT       <number of tuples>]
```

SQL Query: Temporal Relation

- Result of a **SELECT** clause that exists temporally, which assists you in formulating a query

- Syntax:
  SELECT <attributes>
  FROM R1, R2, (SELECT …) <alias>, …, RN
  WHERE <condition>;

- Must always use an alias to denote the result relation of the **SELECT** command

SQL Example: Temporal Relation

Find fname, lname of male employees with salary > 50K

```
SELECT *
FROM    (SELECT fname, lname, salary
         FROM    employee
         WHERE sex = ‘M’) r1
WHERE   r1.salary > 50000
```

SQL Query: Temporal Relation Notes

- You can use multiple temporal relations
- You cannot use a temporal relation to create another temporal relation

Example of incorrect usage:

```
SELECT …
FROM    …,
            (SELECT …)  r1,
            (SELECT … FROM r1  …) r2,
WHERE …;
```

SQL Query: WITH

- SQL-99 standard introduced **WITH** clause to help refine the result of a query (another way to achieve temporal relation)
  - Some vendors do not support **WITH** (e.g., MySQL)
- Syntax:

```
WITH   <alias> AS (SELECT …)[,
       <alias2> AS (SELECT …)]
SELECT <query>;
```

- Can be used to perform “refinement” on a query
  - Subsequent queries in the **WITH** clause can use the results of the previous query

SQL Example: WITH

Find all information on dependents of John Smith

```
WITH r1 as (SELECT *
            FROM employee
            WHERE fname = 'John'
                  AND lname = 'Smith')
SELECT *
FROM   dependent
WHERE  essn IN (SELECT ssn from r1);
```

SQL Query: JOIN Operations

- SQL-99 standard added several join operations:
  - **INNER JOIN** (normal join)
  - **LEFT JOIN** (left outer join)
  - **RIGHT JOIN** (right outer join)
  - **FULL JOIN** (outer join)
- Each operation results in a relation
- Operation can only appear in:
  - **FROM** clause of **SELECT** command
  - **WHERE** clause of **SELECT** command with an operator that uses a sub-query

SQL Query: [INNER] JOIN

- Compute the (inner) join between tables r1 and r2 with a given join condition

- Syntax:
  r1 JOIN r2 ON <join-condition>;
  or
  r1 INNER JOIN r2 ON <join-condition>;

- **JOIN** operator makes the SQL query look a lot like RA query

- Can join more than 2 relations

SQL Example: INNER JOIN

Find fname, lname of employees in the ‘Research’ department

RA Query:

$$\pi_{\text{fname,lname}}(\sigma_{\text{dname='Research'}})(\text{EMPLOYEE} \bowtie_{\text{dno=dnumber}} \text{DEPARTMENT}))$$

SQL Query:

```
SELECT fname, lname
FROM    (employee JOIN department
            ON dno = dnumber)
WHERE  dname = ‘Research’;
```

SQL Query: OUTER JOIN

- Compute the outer join between tables r1 and r2 with a given join condition - see RA slides for details on difference between left, right, and full outer joins

- Syntax:
  r1 LEFT | RIGHT| FULL [OUTER] JOIN r2 on <join condition>;

- Results in NULL values for the attributes where non-matching tuples occur

SQL Query: NATURAL JOIN

- Compute the natural join on attributes with the same names from two or more tables with the common attribute appearing only once in the result

- Syntax:
  r1 NATURAL JOIN r2;

- Example:
  SELECT *
  FROM works_on NATURAL JOIN dependent;

SQL Query: CROSS JOIN

- Cross join is the same as a Cartesian Product

- Syntax:
  r1 CROSS JOIN r2;

- Example:
  SELECT ssn, fname, lname, dno, dnumber, dname
  FROM employee CROSS JOIN dependent;

SQL Outline

- Data definition
  - Database Creation
  - Table Creation
- Query (SELECT)
- Data update (INSERT, DELETE, UPDATE)
- View definition

SQL Modifications/Updates

- A modification command does not return a result but it changes the database

- There are 3 kinds of modifications

- **INSERT** tuple(s)

- **DELETE** tuple(s)

- **UPDATE** the value(s) of existing tuples

SQL Modification: INSERT

- Add one more more tuples to an existing relation
- Two forms of **INSERT**:
  - Literal values (constant or known values)
  - Result from a **SELECT** command

SQL Modification: INSERT (2)

Inserting a tuple using literal/constant values
Syntax:
INSERT INTO <table name>[(<attr names>)]
VALUES (<list of values>);

- Complete tuple: omitting [(<attr names>)] means you must specify all attribute values in the exact order defined in relation

- Partial tuple: specify a subset of the attribute values in the same order as the list of attributes [(<attr names>)]

SQL Modification: INSERT (3)

Inserting a tuple using SELECT command

Syntax:
INSERT INTO <table name>[(<attr names>)] (<SELECT subquery>)

- Multiple tuples may be added dependent on the **SELECT** subquery relation

SQL Example: INSERT

- Complete tuple:

  INSERT INTO employee VALUES (‘Joyce’, ‘C’, ‘Ho’, ‘111223333’, ’1985-02-05’, ‘400 Dowman Drive, Atlanta, GA’, ‘F’, ‘150000’, ‘987654321’, 5);

- Partial tuple:

  INSERT INTO employee(fname, lname, ssn) VALUES (‘Joyce’, ‘Ho’, ‘111223333’);

SQL Example: INSERT w/ SELECT

Suppose we want a new table that has the name, number of employees, and total salaries for each department. We first create the table then load it with the information from the database.

```
CREATE TABLE dept_info
( dept_name   VARCHAR(10),
  no_of_emps  INT,
  tot_salary      INT);

INSERT INTO dept_info
          (SELECT       dname, count(*), sum(salary)
           FROM         department, employee
           WHERE        dnumber = dno
           GROUPY BY dname);
```

MySQL: Bulk Import

- All respectable RDBMS provide utilities to import data from text files
  - Syntax for uploading data will vary based on vendor
- MySQL allows the **LOAD DATA INFILE** (http://dev.mysql.com/doc/refman/5.7/en/load-data.html)
  - For a pipe-delimited file (| separates each column):
    LOAD DATA LOCAL INFILE <filename>
    {REPLACE | IGNORE} INTO TABLE <table name>
    FIELDS TERMINATED BY '|';

SQL Modification: DELETE

- Remove tuples from a relation

- Syntax:
  DELETE FROM <relation>
  WHERE <condition>;

- Be careful! All tuples that satisfy the condition clause are deleted

- Tuples are deleted from only one table at a time unless **CASCADE** is specified on a referential integrity constraint

- What happens if we don't specify a **WHERE** clause?

SQL Example: DELETE

Delete all employees with the last name Brown

```
DELETE FROM employee
WHERE  lname = 'Brown';
```

SQL Example: DELETE (2)

Delete all employees from the ‘Research’ department who have more than 2 dependents

```
DELETE FROM employee
WHERE  dno IN (SELECT dnumber
               FROM    department
               WHERE  dname = ‘Research’)
AND  ssn  IN (SELECT essn
               FROM    dependent
               GROUP BY  essn
               HAVING  COUNT(name) > 2);
```

SQL Modification: UPDATE

- Modify/change certain attributes in certain tuples of a relation

- Syntax:
  UPDATE <relation>
  SET <list of attribute assignments>
  WHERE <condition>;

- **UPDATE** command modifies tuples in the same relation

SQL Example: UPDATE

Change the location and controlling department number of project 10 to ‘Bellaire’ and 5, respectively.

```
UPDATE project
SET    plocation = ‘Bellaire’, dnum = 5
WHERE  pnumber = 10;
```

SQL Example: UPDATE (2)

Give all employees in the ‘Research’ department a 10% raise

```
UPDATE employee
SET    salary = salary * 1.1
WHERE  dno IN (SELECT dnumber
               FROM   department
               WHERE dname = ‘Research’);
```

- Reference to salary attribute on the right of = refers to the salary value before modification
- Reference to salary attribute on the left of = refers to salary value after modification

SQL: VIEW

- A view is a virtual table, a relation that is defined in terms of the contents of other tables and views

- A view d*oes not* exist in the physical form

- In contrast, a relation whose value is really in the database is called a base table

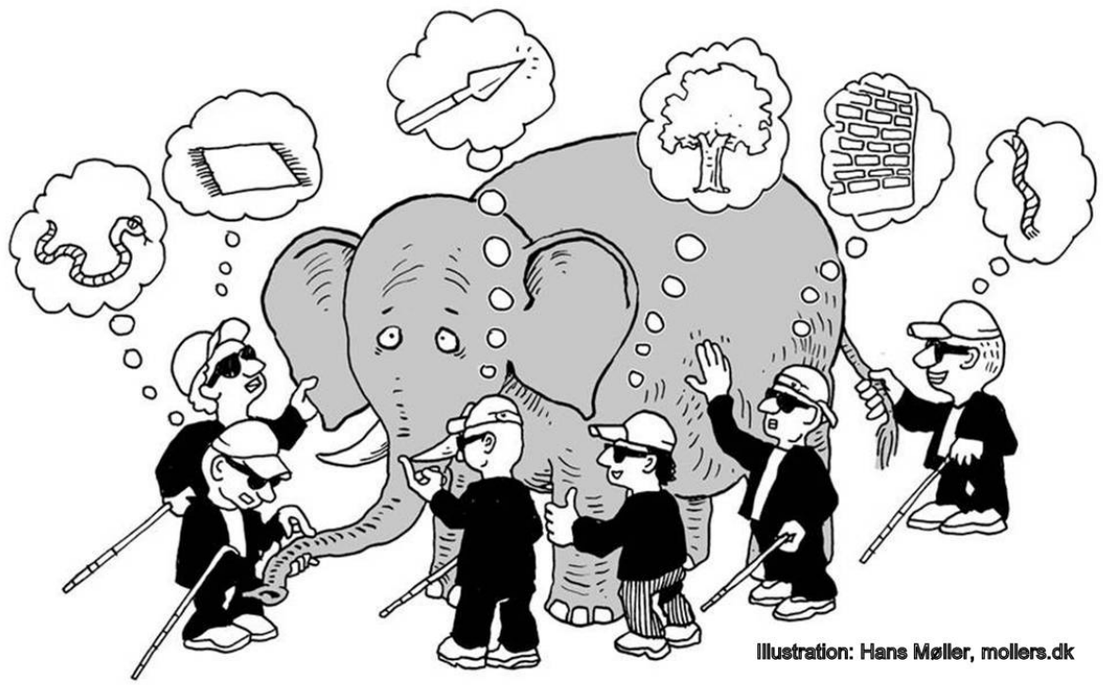
- Syntax:
  CREATE VIEW <name> AS <query>;

SQL: View & Logical Data Independence

- Recall Logical Data Independence
  (class on Database Concepts)
    - Ability to present the stored information in a different way to different users

- View can be adapted to the need of the user

- If conceptual schema changes, only the **SELECT** query needed to construct view needs to change



SQL Example: VIEW

- Suppose an administrator maintains a list of activities of all employees which contains the following information:
  fname, lname, project_name, hours_worked

- Regular SELECT query:

```
SELECT fname, lname, pname, hours
FROM    employee, works_on, project
WHERE  ssn = essn AND pno = pnumber;
```

- Create VIEW for the admin:

```
CREATE VIEW emp_activity
AS (SELECT fname, lname, pname, hours
      FROM    employee, works_on, project
      WHERE  ssn = essn AND pno = pnumber);
```

SQL: VIEW Advantages

- View can be used in queries like an ordinary relation
  - When a view is used in a **SELECT** query, the virtual relation is computed first

- Simplify complex queries by hiding them from the end-user and applications

- Limit data access to specific users (expose only non-sensitive data) and provides extra security for read/write access

- Enables backward compatibility - changes to database won't affect changes to other applications

SQL: VIEW Disadvantages

- Querying data from database view can be slow (since view is computed each time)

- Tables dependency - updates to the underlying tables will force changes to the view itself to make it work properly

- Most data manipulation statements (**INSERT**, **DELETE**, **UPDATE**) are not possible on the view

SQL Data Update & View: Recap

- Query
  - Temporal Relation / WITH
  - JOIN
- SQL Modification
  - INSERT
  - DELETE
  - UPDATE
- SQL Views

More SQL Practice

Find the fname, lname of employees with more than 2 dependents and work on all projects controlled by department #1

More SQL Practice

Find the fname, lname of employees with more than 2 dependents and work on all projects controlled by department #1

```
SELECT fname, lname
FROM   employee
WHERE <employee has more than 2 dependents>
   AND  <works on all projects controlled by dept #1>;
```

First formulate in words

conquer each subquery separately

More SQL Practice: Subquery #1

Find employee that has more than 2 dependents

```
SELECT essn
FROM   dependent
GROUP BY essn
HAVING  COUNT(name) > 2;
```

More SQL Practice: Subquery #2

Find employees that works on all projects controlled by department #1 - set difference technique

```
SELECT ssn
FROM   employee e
WHERE  NOT EXISTS
       < set of projects controlled by department #1 >
       - <set of projects worked on by e.ssn>;
```

More SQL Practice: Subquery #2

Find employees that works on all projects controlled by department #1 - set difference technique

```
SELECT ssn
FROM    employee e
WHERE NOT EXISTS (SELECT pnumber
                  FROM   project
                  WHERE pnumber IN
                        ( SELECT pnumber
                          FROM    project
                          WHERE  dnum = 1)
                    AND pnumber NOT IN
                        ( SELECT pno
                          FROM    works_on
                          WHERE essn = e.ssn));
```

More SQL Practice: Putting it Together

Find the fname, lname of employees with more than 2 dependents and work on all projects controlled by department #1

```
SELECT fname, lname
FROM    employee
WHERE  ssn IN ( SELECT essn
                FROM    dependent
                GROUP BY essn
                HAVING  COUNT(name) > 2)
AND    ssn IN ( SELECT ssn
                FROM    employee e
                WHERE NOT EXISTS (SELECT pnumber
                                  FROM   project
                                  WHERE pnumber IN
                                      (SELECT pnumber
                                       FROM    project
                                       WHERE  dnum = 1)
                                     AND pnumber NOT IN
                                       ( SELECT pno
                                         FROM    works_on
                                         WHERE essn = e.ssn)));
```

More SQL Practice (2)

Find the department name, and the number of employees in that department that earns more than 40K for departments with at least 2 employees

More SQL Practice (2)

Find the department name, and the number of employees in that department that earns more than 40K for departments with at least 2 employees

```
SELECT    dname, COUNT(ssn)
FROM      department, employee
WHERE     dnumber = dno
  AND     salary > 40000
GROUP BY dname
HAVING    COUNT(ssn) >= 2;
```

What is wrong with this solution?

More SQL Practice (2): Solution

Find the department name, and the number of employees in that department that earns more than 40K for departments with at least 2 employees

```
SELECT      dname, COUNT(ssn)
FROM        department, employee
WHERE       dnumber = dno
    AND     salary > 40000
    AND     dno IN ( SELECT     dno
                     FROM       employee
                     GROUP BY dno
                     HAVING     COUNT(ssn) >= 2 )
GROUP BY dname;
```

More SQL Practice (3)

Find fname, name of employees who work on 2 or more projects together with John Smith

More SQL Practice (4)

Find departments who have 2 or more employees working on all projects controlled by ‘Research’ department