CSC 742
Database Management Systems

Topic #10: SQL

Part A: Data Definition Language (DDL)

Schema and Catalog

- Schema
  - A collection of relations (tables)
  - Identified by a schema name
  - Include tables, constraints, views, domains, and others.

CREATE SCHEMA COMPANY AUTHORIZATION Bob;

- Catalog
  - A named collection of schemas.
  - Integrity constraints can be defined between relations only if they exist in the same catalog.

CREATE TABLE EMPLOYEE
  (FNAME VARCHAR(15) NOT NULL,
   LNAME VARCHAR(15) NOT NULL,
   SSN CHAR(9) NOT NULL,
   ...,
   SUPERSSN CHAR(9),
   DN INT NOT NULL,
   PRIMARY KEY (SSN),
   FOREIGN KEY (SUPERSSN) REFERENCES EMPLOYEE(SSN),
   FOREIGN KEY (DN) REFERENCES DEPARTMENT(DNUMBER));

CREATE TABLE DEPARTMENT
  (DNAME VARCHAR(15) NOT NULL,
   DNUMBER INT NOT NULL,
   MGRSSN CHAR(9) NOT NULL,
   MGRSTARTDATE DATE,
   PRIMARY KEY(DNUMBER),
   UNIQUE (DNAME),
   FOREIGN KEY (MGRSSN) REFERENCES EMPLOYEE(SSN));

CREATE TABLE DEPARTMENT
  (DNAME VARCHAR(15) NOT NULL,
   DNUMBER INT NOT NULL,
   MGRSSN CHAR(9) NOT NULL,
   MGRSTARTDATE DATE,
   PRIMARY KEY(DNUMBER),
   FOREIGN KEY (MGRSSN) REFERENCES EMPLOYEE(SSN));

Referential Triggered Actions

- Two events
  - ON DELETE
  - ON UPDATE

- Three options
  - SET NULL
  - SET DEFAULT
  - CASCADE
Self-Study

- DROP SCHEMA
- DROP TABLE
- ALTER TABLE
  - Alter attributes
  - Alter constraints

---

Part B: Data Manipulation Language (DML)

DML

- Our focus is how to formulate queries.
- Self-study:
  - INSERT
  - DELETE
  - UPDATE

---

SELECT

- Used for retrieval
- Used to specify subqueries for retrieval and for the other operations
- Not quite the $\sigma$ or select operator of the relational algebra
- SELECT
  - tuple queries
  - aggregate queries

---

SELECT (Cont’d)

- Basic paradigm
  - SELECT column$_1$, …, column$_n$
  - FROM table$_1$, …, table$_m$
  - WHERE condition
  - The WHERE condition can be a boolean combination of other conditions involving the tables table$_1$ through table$_m$

---

Employee

<table>
<thead>
<tr>
<th>SSN</th>
<th>Lname</th>
<th>Fname</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>111-22-3333</td>
<td>Smith</td>
<td>John</td>
<td>30000</td>
</tr>
<tr>
<td>121-23-3333</td>
<td>Wong</td>
<td>Frank</td>
<td>45000</td>
</tr>
<tr>
<td>133-32-1342</td>
<td>Wallace</td>
<td>Jennifer</td>
<td>43000</td>
</tr>
<tr>
<td>154-33-3333</td>
<td>Borg</td>
<td>James</td>
<td>56000</td>
</tr>
<tr>
<td>555-44-5555</td>
<td>English</td>
<td>Joyce</td>
<td>53000</td>
</tr>
</tbody>
</table>

List the names of the employees whose salary is more than 50,000.

SELECT Lname, Fname
FROM Employee
WHERE Salary > 50000;

Relational Algebra:
Select + Project
List the names of the dependents of Alice Zelaya.

```
SELECT Dependent.Fname, Dependent.Lname
FROM Employee, Dependent
WHERE Employee.Fname = 'Alice' AND
Employee.Lname = 'Zelaya' AND
Employee.SSN = Dependent.ESSN;
```

Relational Algebra:
Select + Project + Join

---

**Exercise 1.**

- Find names of employees in the research dept

```
Employee(Fname, Lname, SSN, Bdate, Address, Sex, Salary, SuperSSN, Dno)

SELECT __________, __________
FROM __________, __________
WHERE __________, __________;
```

---

**SQL Variants:**

* Employee

<table>
<thead>
<tr>
<th>SSN</th>
<th>Lname</th>
<th>Fname</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>111-22-3333</td>
<td>Smith</td>
<td>John</td>
<td>30000</td>
</tr>
<tr>
<td>121-23-3333</td>
<td>Wong</td>
<td>Frank</td>
<td>45000</td>
</tr>
<tr>
<td>153-32-1342</td>
<td>Wallace</td>
<td>Jennifer</td>
<td>43000</td>
</tr>
<tr>
<td>154-33-3333</td>
<td>Borg</td>
<td>James</td>
<td>53000</td>
</tr>
<tr>
<td>555-44-5555</td>
<td>English</td>
<td>Joyce</td>
<td>53000</td>
</tr>
</tbody>
</table>

```
SELECT *
FROM Employee
WHERE Salary > 50000;
```

---

**SQL Variants:**

ALL and DISTINCT

```
SELECT ALL, Salary
FROM Employee;
```

---

**SQL Variants:**

Renaming

```
Employee(Fname, Lname, SSN, Bdate, Address, Sex, Salary, SuperSSN, Dno)

SELECT E.Fname, E.Lname, S.Fname, S.Lname
FROM Employee AS E, Employee AS S
WHERE E.SuperSSN = S.SSN;
```

---

**Exercise 2**

- For every project in 'Stafford' list the controlling dept number and the dept manager's last name

```
Employee(Fname, Lname, SSN, Bdate, Address, Sex, Salary, SuperSSN, Dno)
Department(Dname, Dnumber, MgrSSN, MgrStartDate)
Project(Pname, Pnumber, Plocation, Dnum)

SELECT __________, __________
FROM __________, __________, __________
WHERE __________, __________ AND
_______________;
```

---

**SQL Variants:**

```
SELECT *
FROM Employee
WHERE Salary > 50000;
```
**SQL Variants: IS [NOT] NULL**

<table>
<thead>
<tr>
<th>Employee</th>
<th>SSN</th>
<th>Lname</th>
<th>Fname</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>111-22-333</td>
<td>Smith</td>
<td>John</td>
<td>30000</td>
</tr>
<tr>
<td></td>
<td>121-23-333</td>
<td>Wong</td>
<td>Frank</td>
<td>30000</td>
</tr>
<tr>
<td></td>
<td>153-32-1342</td>
<td>Wallace</td>
<td>Jennifer</td>
<td>43000</td>
</tr>
<tr>
<td></td>
<td>154-33-333</td>
<td>Borg</td>
<td>James</td>
<td>NULL</td>
</tr>
<tr>
<td></td>
<td>555-44-5555</td>
<td>English</td>
<td>Joyce</td>
<td>53000</td>
</tr>
</tbody>
</table>

**SQL Variants: ORDER BY**

<table>
<thead>
<tr>
<th>Employee</th>
<th>SSN</th>
<th>Lname</th>
<th>Fname</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>111-22-333</td>
<td>Smith</td>
<td>John</td>
<td>30000</td>
</tr>
<tr>
<td></td>
<td>121-23-333</td>
<td>Wong</td>
<td>Frank</td>
<td>30000</td>
</tr>
<tr>
<td></td>
<td>153-32-1342</td>
<td>Wallace</td>
<td>Jennifer</td>
<td>43000</td>
</tr>
<tr>
<td></td>
<td>154-33-333</td>
<td>Borg</td>
<td>James</td>
<td>NULL</td>
</tr>
<tr>
<td></td>
<td>555-44-5555</td>
<td>English</td>
<td>Joyce</td>
<td>53000</td>
</tr>
</tbody>
</table>

**Nested Queries**

- A nested query (subquery): a query in the WHERE clause of another query.
- Some paradigms of subqueries are:
  - `<column> [NOT] IN <subquery>`
  - `<column> <op> <subquery>`
  - `<column> <op> ANY|ALL <subquery>`
  - `[NOT] EXISTS <subquery>`
  - `<subquery> CONTAINS <subquery>`

**Nested Queries Example: 1**

- Find all the dept 6 projects located where a dept 5 project is located

```sql
SELECT Project.Pnumber FROM Project WHERE ______ AND Project.Plocation IN (SELECT Project.Plocation FROM Project WHERE ______);
```

**Nested Queries Example: 2**

- List the name of the employee who has the highest salary.

```sql
SELECT E.Lname, E.Fname FROM Employee WHERE 
    (SELECT Salary FROM Employee);
```

**Subqueries**

- Subqueries can't modify tables
- For `<column> <op> <subquery>`,
  `<subquery>` can return exactly one value from one column
- **Correlated subquery**: When the subquery includes a field that is supplied from the outer query
Correlated Subqueries: 1

Find names of employees who live where their dept is located
Taking a slight liberty with the address column

```
SELECT E.Lname, E.Fname
FROM Employee AS E
WHERE (SELECT D.Dno
      FROM Department AS D
      WHERE D.Location = ____) ;
```

Correlated Subqueries: 2

- These are the most interesting kinds of subqueries
- Intuitively, the subquery is evaluated once for each tuple in the main query
- Often the same table will show up in both the main query and the subquery
- Here, the use of aliases is a necessity, not merely a convenience

CONTAINS Example

Find SSNs of employees who work on all the projects controlled by dept 5.

```
SELECT Fname, Lname
FROM Employee AS E
WHERE (SELECT Pnum
       FROM Project AS P
       WHERE (SELECT W.no
              FROM Works_on AS W
              WHERE _ _ _ _ _ _ _ )
      CONTAINS
       (SELECT Pnum
        FROM Project AS P
        WHERE _ _ _ _ _ _ _ )) ;
```

Achtung!

- ANY means some
- Thus, "salary > ANY (...)" could hold for the second lowest salary from the subquery evaluated within the ANY

EXISTS

- Check whether the result of a subquery is empty.
- List the names of managers who have at least one dependent.

```
SELECT Fname, Lname
FROM Employee AS E
WHERE EXISTS (SELECT *
              FROM Dependent AS D WHERE ____) AND EXISTS (SELECT *
                                                          FROM Department AS D WHERE ____) ;
```
Aggregate Operators

- SQL includes operators that combine data from a single column of several tuples
  - SUM (only numeric)
  - AVG (only numeric)
  - COUNT
  - MAX and MIN
- All eliminate NULLs except COUNT(*)
- All include duplicates unless DISTINCT
- COUNT(*) includes NULLs and duplicates

List the sum of the salaries of all employees, the highest, the lowest, and the average salary:

<table>
<thead>
<tr>
<th>SSN</th>
<th>Lastname</th>
<th>Fname</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>111-22-3333</td>
<td>Smith</td>
<td>John</td>
<td>80000</td>
</tr>
<tr>
<td>121-23-3333</td>
<td>Wong</td>
<td>Frank</td>
<td>80000</td>
</tr>
<tr>
<td>153-32-1342</td>
<td>Wallace</td>
<td>Jennifer</td>
<td>43000</td>
</tr>
<tr>
<td>154-33-3333</td>
<td>Borg</td>
<td>James</td>
<td>56000</td>
</tr>
<tr>
<td>555-44-5555</td>
<td>English</td>
<td>Joyce</td>
<td>53000</td>
</tr>
</tbody>
</table>

SELECT SUM(Salary), MAX(Salary), MIN(Salary), AVG(Salary) FROM Employee;

GROUP BY

- SELECT ...
- FROM ...
- WHERE ...
- GROUP BY columns

- The GROUP BY columns must appear in the SELECT list as well.

List the highest salary, the lowest, and the average salary of each department:

<table>
<thead>
<tr>
<th>SSN</th>
<th>Lastname</th>
<th>Fname</th>
<th>DNo</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>111-22-3333</td>
<td>Smith</td>
<td>John</td>
<td>1</td>
<td>80000</td>
</tr>
<tr>
<td>121-23-3333</td>
<td>Wong</td>
<td>Frank</td>
<td>1</td>
<td>80000</td>
</tr>
<tr>
<td>153-32-1342</td>
<td>Wallace</td>
<td>Jennifer</td>
<td>2</td>
<td>43000</td>
</tr>
<tr>
<td>154-33-3333</td>
<td>Borg</td>
<td>James</td>
<td>2</td>
<td>56000</td>
</tr>
<tr>
<td>555-44-5555</td>
<td>English</td>
<td>Joyce</td>
<td>1</td>
<td>53000</td>
</tr>
</tbody>
</table>

SELECT DNo, MAX(Salary), MIN(Salary), AVG(Salary) FROM Employee GROUP BY Dno;

HAVING

- A selection condition that applies to different groups resulting from a GROUP BY.

List the highest salary, the lowest, and the average salary of the departments whose numbers are less than 3:

<table>
<thead>
<tr>
<th>SSN</th>
<th>Lastname</th>
<th>Fname</th>
<th>DNo</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>111-22-3333</td>
<td>Smith</td>
<td>John</td>
<td>1</td>
<td>80000</td>
</tr>
<tr>
<td>121-23-3333</td>
<td>Wong</td>
<td>Frank</td>
<td>1</td>
<td>80000</td>
</tr>
<tr>
<td>153-32-1342</td>
<td>Wallace</td>
<td>Jennifer</td>
<td>2</td>
<td>43000</td>
</tr>
<tr>
<td>154-33-3333</td>
<td>Borg</td>
<td>James</td>
<td>2</td>
<td>56000</td>
</tr>
<tr>
<td>555-44-5555</td>
<td>English</td>
<td>Joyce</td>
<td>1</td>
<td>53000</td>
</tr>
</tbody>
</table>

SELECT Dno, MAX(Salary), MIN(Salary), AVG(Salary) FROM Employee GROUP BY Dno HAVING Dno < 3;
Calculus to SQL: 1
A methodology for producing sound SQL queries
- Write calculus expressions
- Systematically map them to SQL queries
  - "normalize" the queries
  - Map $\forall$ to $\exists$
  - map free variables
  - map quantifiers

Find SSNs of employees who work on all the projects controlled by dept 5.

Employee(Fname, Lname, SSN, Bdate, Address, Sex, Salary, SuperSSN, Dno)

Works_on(ESSN, Pro, Hours)

Project(ProjName, Pnum, Project, Dno)

W orks _ on ( E S S N , P n o , H o u r s )
\[
\{ e . S S N \mid \text{Employee}(e) \land (\forall p : \text{Project}(p) \land (p . D n u m \neq 5 \lor (\exists w : \text{W orks}_o n (w) \land w . P n o = p . P n u m b e r \land w . E S S N = e . S S N ))) \}
\]

¬ (A \land B) \iff (\neg A \lor \neg B)

¬ (A \lor B) \iff (\neg A \land \neg B)

SQL Views
- An SQL view is a table derived from other tables
  - base (physical) tables—ultimately depend on these defining tables
  - other views
  - Views are
    - usually virtual
    - sometimes materialized

View Specification
- Views are specified with the paradigm
  CREATE VIEW view
  AS SELECT ...
- The SELECT ... part is the defining query
- In a view definition, we can
  - define column names of view
  - use aggregation (GROUP BY)
View Resolution

- Views are
  - computed by a sort of macro expansion when needed—view resolution
  - query modification: compute fresh
  - view materialization: store
  - always up to date: modifications to the defining tables are automatically reflected in the view

General Constraints

- SQL allows declarative constraints such as
  - CREATE ASSERTION assertion-name
    - CHECK (enhanced subselect query)
- The DBMS checks if any ASSERTION is ever violated