Carnegie Mellon Univ.
Dept. of Computer Science
15-415/615 - DB Applications

C. Faloutsos – A. Pavlo
Lecture#7: Fun with SQL (Part 2)

Administrivia

• HW2 is due next Monday.

Last Class

• SELECT/INSERT/UPDATE/DELETE
• Table Definition (DDL)
• NULLs
• String/Date/Time/Set/Bag Operations
• Output Redirection/Control
• Aggregates/Group By

Today's Jam

• Complex Joins
• Views
• Nested Queries
• Common Table Expressions
• Triggers
• Database Application Example
Example Database

**STUDENT**

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Trump</td>
<td>trump@cs</td>
<td>45</td>
<td>4.0</td>
</tr>
<tr>
<td>53688</td>
<td>Bieber</td>
<td>jbieber@cs</td>
<td>21</td>
<td>3.9</td>
</tr>
<tr>
<td>53655</td>
<td>Tupac</td>
<td>shakur@cs</td>
<td>26</td>
<td>3.5</td>
</tr>
</tbody>
</table>

**ENROLLED**

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Pilates101</td>
<td>C</td>
</tr>
<tr>
<td>53688</td>
<td>Reggae203</td>
<td>D</td>
</tr>
<tr>
<td>53688</td>
<td>Topology112</td>
<td>A</td>
</tr>
<tr>
<td>53666</td>
<td>Massage105</td>
<td>D</td>
</tr>
</tbody>
</table>

**COURSE**

<table>
<thead>
<tr>
<th>cid</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pilates101</td>
<td>Pilates</td>
</tr>
<tr>
<td>Reggae203</td>
<td>20th Century Reggae</td>
</tr>
<tr>
<td>Topology112</td>
<td>Topology + Squirrels</td>
</tr>
<tr>
<td>Massage105</td>
<td>Massage &amp; Holistic Therapy</td>
</tr>
</tbody>
</table>

Join Query Grammar

- **Join-Type**: The type of join to compute.
- **Qualification**: Expression that determines whether a tuple from table1 can be joined with table2. Comparison of attributes or constants using operators $=, \neq, <, >, \leq, \text{and} \geq$.

**SELECT** ... 
**FROM** table-name1 join-type table-name2 
**ON** qualification 
[**WHERE** ...]

**INNER JOIN**

**SELECT** name, cid, grade 
**FROM** student, enrolled 
**WHERE** student.sid = enrolled.sid

**OUTER JOIN**

**SELECT** name, cid, grade 
**FROM** student **LEFT OUTER JOIN** enrolled 
**ON** student.sid = enrolled.sid

**SELECT** name, cid, grade 
**FROM** student, enrolled 
**ON** student.sid = enrolled.sid

<table>
<thead>
<tr>
<th>name</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bieber</td>
<td>Reggae203</td>
<td>D</td>
</tr>
<tr>
<td>Bieber</td>
<td>Topology112</td>
<td>A</td>
</tr>
<tr>
<td>Trump</td>
<td>Massage105</td>
<td>D</td>
</tr>
<tr>
<td>Trump</td>
<td>Pilates101</td>
<td>C</td>
</tr>
<tr>
<td>Tupac</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>
### OUTER JOIN

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Trump</td>
<td>trump@cs</td>
<td>45</td>
<td>4.0</td>
</tr>
<tr>
<td>53688</td>
<td>Bieber</td>
<td>jbieber@cs</td>
<td>21</td>
<td>3.9</td>
</tr>
<tr>
<td>53677</td>
<td>Tupac</td>
<td>shakur@cs</td>
<td>26</td>
<td>3.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Pilates101</td>
<td>C</td>
</tr>
<tr>
<td>53688</td>
<td>Reggae203</td>
<td>D</td>
</tr>
<tr>
<td>53688</td>
<td>Topology112</td>
<td>A</td>
</tr>
<tr>
<td>53666</td>
<td>Massage105</td>
<td>D</td>
</tr>
</tbody>
</table>

### JOIN TYPES

- **INNER JOIN**: Join where A and B have same value
- **LEFT OUTER JOIN**: Join where A and B have same value AND where only A has a value
- **RIGHT OUTER JOIN**: Join where A and B have same value AND where only B has a value
- **FULL OUTER JOIN**: Join where A and B have same value AND where A or B have unique values
- **CROSS JOIN**: Cartesian Product

### TODAY'S JAM

- Complex Joins
- Views
- Nested Queries
- Common Table Expressions
- Triggers
- Database Application Example
Views

- Creates a “virtual” table containing the output from a **SELECT** query.
- Mechanism for hiding data from view of certain users.
- Can be used to simplify a complex query that is executed often.
  - Won’t make it faster though!

View Example

- Create a view of the CS student records with just their id, name, and login.

```sql
CREATE VIEW CompSciStudentInfo AS
SELECT sid, name, login
FROM student
WHERE login LIKE '%@cs';
```

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Trump</td>
<td>trump@cs</td>
<td>45</td>
<td>4.0</td>
</tr>
<tr>
<td>53688</td>
<td>Bieber</td>
<td>jbieber@cs</td>
<td>21</td>
<td>3.9</td>
</tr>
</tbody>
</table>

Original Table

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Trump</td>
<td>trump@cs</td>
<td>45</td>
<td>4.0</td>
</tr>
<tr>
<td>53688</td>
<td>Bieber</td>
<td>jbieber@cs</td>
<td>21</td>
<td>3.9</td>
</tr>
</tbody>
</table>

View

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>Trump</td>
<td>trump@cs</td>
</tr>
<tr>
<td>53688</td>
<td>Bieber</td>
<td>jbieber@cs</td>
</tr>
</tbody>
</table>

View Example

- Create a view with the average age of the students enrolled in each course.

```sql
CREATE VIEW CourseAge AS
SELECT cid, AVG(age) AS avg_age
FROM student, enrolled
WHERE student.sid = enrolled.sid
GROUP BY enrolled.cid;
```

<table>
<thead>
<tr>
<th>cid</th>
<th>avg_age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Massage105</td>
<td>45.0</td>
</tr>
<tr>
<td>Pilates101</td>
<td>45.0</td>
</tr>
<tr>
<td>Topology112</td>
<td>21.0</td>
</tr>
<tr>
<td>Reggae203</td>
<td>21.0</td>
</tr>
</tbody>
</table>

Views vs. SELECT INTO

- INTO→Creates static table that does not get updated when student gets updated.
- VIEW→Dynamic results are only materialized when needed.
Materialized Views

- Creates a view containing the output from a SELECT query that is automatically updated when the underlying tables change.

```
CREATE MATERIALIZED VIEW AvgGPA AS
  SELECT AVG(gpa) AS avg_gpa
  FROM student
  WHERE login LIKE '%@cs'
```

Today's Jam

- Complex Joins
- Views
- Nested Queries
- Common Table Expressions
- Triggers
- Database Application Example

Nested Queries

- Queries containing other queries
- Inner query:
  - Can appear in FROM or WHERE clause

```
SELECT cname FROM customer
WHERE acctno IN (SELECT acctno FROM account)
```

Think of this as a function that returns the result of the inner query

```
<table>
<thead>
<tr>
<th>cname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Johnson</td>
</tr>
<tr>
<td>Smith</td>
</tr>
<tr>
<td>Jones</td>
</tr>
<tr>
<td>Smith</td>
</tr>
</tbody>
</table>
```

Nested Queries

- Find the names of students in ‘Massage105’

```
SELECT name FROM student
WHERE ...
```

“sid in the set of people that take Massage105”
Nested Queries

- **Find the names of students in ‘Massage105’**

```sql
SELECT name FROM student
WHERE ... 
    SELECT sid FROM enrolled
    WHERE cid = 'Massage105'
```

- **ALL** → Must satisfy expression for all rows in sub-query
- **ANY** → Must satisfy expression for at least one row in sub-query.
- **IN** → Equivalent to ‘=ANY()’.
- **EXISTS** → At least one row is returned.

- Nested queries are difficult to optimize. Try to avoid them if possible.
Nested Queries

• Find student record with the highest id.
• This won’t work in **SQL-92**:

```
SELECT MAX(sid), name FROM student;
```

X

• Runs in **MySQL**, but you get wrong answer:

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>53688</td>
<td>Tupac</td>
</tr>
</tbody>
</table>

“is greater than every other sid”

```sql
SELECT sid, name FROM student
WHERE sid => ALL(
   SELECT sid FROM enrolled
)
```

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>53688</td>
<td>Bieber</td>
</tr>
</tbody>
</table>
Nested Queries

- Find student record with the highest id.

```sql
SELECT sid, name FROM student
WHERE sid IN (SELECT MAX(sid) FROM enrolled)
```

- Find all courses that nobody is enrolled in.

```sql
SELECT * FROM course
WHERE NOT EXISTS(
    SELECT * FROM enrolled
    WHERE course.cid = enrolled.cid
)
```

- Find all courses that nobody is enrolled in.

```sql
SELECT * FROM course
WHERE NOT EXISTS(
    SELECT * FROM enrolled
    WHERE course.cid = enrolled.cid
)
```
Today's Jam

- Complex Joins
- Views
- Nested Queries
- Common Table Expressions
- Window Functions
- Triggers
- Database Application Example

Common Table Expressions

- Provides a way to write auxiliary statements for use in a larger query.
- Alternative to nested queries and views.

```sql
WITH cteName AS ( 
   SELECT 1 
)  
SELECT * FROM cteName
```

- Find student record with the highest id that is enrolled in at least one course.

```sql
WITH cteSource (maxId) AS ( 
   SELECT MAX(sid) FROM enrolled 
)  
SELECT name FROM student, cteSource 
WHERE student.sid = cteSource.maxId
```

CTEs – Recursion

- Print 1 to 10.

```sql
WITH RECURSIVE cteSource (counter) AS ( 
   (SELECT 1 
    UNION ALL 
    (SELECT counter + 1 FROM cteSource 
     WHERE counter < 10) 
)  
SELECT * FROM cteSource
```

- **Postgres CTE Demo!**
Today's Jam

- Complex Joins
- Views
- Nested Queries
- Common Table Expressions
- Triggers
- Database Application Example

Database Triggers

- Procedural code that is automatically executed in response to certain events on a particular table or view in a database.
- **BEFORE/AFTER**
  - INSERT
  - UPDATE
  - DELETE

Trigger Example

- Set a timestamp field whenever a row in the enrolled table is updated.
- First we need to add our timestamp field.

```sql
ALTER TABLE enrolled
ADD COLUMN updated TIMESTAMP;
```

Trigger Example

- Register a function that sets the ‘updated’ column with the current timestamp.

```sql
CREATE OR REPLACE FUNCTION update_col()
RETURNS TRIGGER AS $$
BEGIN
    NEW.updated = NOW();
    RETURN NEW;
END;
$$ language 'plpgsql';
```

Postgres
Trigger Example

- Invoke the `update_col` function when a row in the enrolled table is updated.

```sql
CREATE TRIGGER update_enrolled_modtime AFTER UPDATE ON enrolled FOR EACH ROW EXECUTE PROCEDURE update_col();
```

MySQL Alternative

- Non-standard way to do this just for setting timestamps.

```sql
CREATE TABLE enrolled (    updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP );
```

Today's Party

- DDLs
- Complex Joins
- Views
- Nested Subqueries
- Triggers
- Database Application Example

Outline of an DB application

- Establish connection with DB server
- Authenticate (user/password)
- Execute SQL statement(s)
- Process results
- Close connection
Database Connection Libraries

• DBMS-specific Libraries
• “Universal” Libraries
  – Open Database Connectivity (ODBC)
  – Java Database Connectivity (JDBC)
• Application framework libraries

ORM Libraries

• Object-Relational Mapping
• Automatically convert classes into database-backed objects.
• Method calls on objects are automatically converted into SQL queries.
• Removes the tediousness of writing SQL queries directly in application code.

ORM Example

```python
class Location(models.Model):
    zipcode = CharField(max_length=5, primary_key=True)
    state = USStateField()
    city = CharField(max_length=64)

class Company(models.Model):
    name = CharField(max_length=64, unique=True)
    address1 = CharField(max_length=128)
    location = ForeignKey(Location)
    website = URLField()
    public = BooleanField(default=True)
```

```sql
CREATE TABLE location (  
  zipcode VARCHAR(5) NOT NULL,  
  state CHAR(2) NOT NULL,  
  city VARCHAR(64) NOT NULL,  
  PRIMARY KEY (zipcode),  
)
CREATE TABLE company (  
  id INT(11) NOT NULL AUTO_INCREMENT,  
  name VARCHAR(64) NOT NULL,  
  address1 VARCHAR(128) NOT NULL,  
  location_id VARCHAR(5) NOT NULL 
    REFERENCES location (zipcode),  
  website VARCHAR(200) NOT NULL,  
  public TINYINT(1) NOT NULL,  
  PRIMARY KEY (id),  
);```
ORM Libraries

- Standalone:
  - Hibernate (Java)
  - SQLAlchemy (Python)
  - Doctrine (PHP)

- Integrated:
  - Django (Python)
  - ActiveRecord (Ruby on Rails)
  - CakePHP (PHP)

Next Class

- We begin discussing storage internals.
- This material will be important for helping you pick up dates at parties.