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

C. Faloutsos – A. Pavlo
Lecture#6: Fun with SQL (Part 1)

Administrivia

• HW1 is due today.
• HW2 is out.

Homework #2: Bike-Share Data

• For each question, generate a SQL query that computes the answer.
  – It will test automatically when you submit.
  – Column names are not important but order is
• You can use Postgres on your laptop or on one of the Andrews machines.
  – Check the “Grade Center” on Blackboard for your machine and port number.

Relational Languages

• A major strength of the relational model: supports simple, powerful querying of data.
• User only needs to specify the answer that they want, not how to compute it.
• The DBMS is responsible for efficient evaluation of the query.
  – Query optimizer: re-orders operations and generates query plan
Relational Languages

- Standardized DML/DDL
  - DML → Data Manipulation Language
  - DDL → Data Definition Language
- Also includes:
  - View definition
  - Integrity & Referential Constraints
  - Transactions

History

- Originally “SEQUEL” from IBM’s System R prototype.
  - Structured English Query Language
  - Adopted by Oracle in the 1970s.
  - Structured Query Language

- Current standard is SQL:2011
  - SQL:2011 → Temporal DBs, Pipelined DML
  - SQL:2008 → TRUNCATE, Fancy ORDER
  - SQL:2003 → XML, windows, sequences, auto-generated IDs.
  - SQL:1999 → Regex, triggers, OO
- Most DBMSs at least support SQL-92
- System Comparison:
  - http://troels.arvin.dk/db/rdbms/

Today's Class: OLTP

- Basic Queries
- Table Definition (DDL)
- NULLs
- String/Date/Time/Set/Bag Operations
- Output Redirection/Control
### Example Database

<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>Kayne</td>
<td>kayne@cs</td>
<td>39</td>
<td>4.0</td>
</tr>
<tr>
<td>53688</td>
<td>Bieber</td>
<td>jbieber@cs</td>
<td>22</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>

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
<td>15-415</td>
<td>C</td>
</tr>
<tr>
<td>53688</td>
<td>15-721</td>
<td>A</td>
</tr>
<tr>
<td>53688</td>
<td>15-826</td>
<td>B</td>
</tr>
<tr>
<td>53655</td>
<td>15-415</td>
<td>C</td>
</tr>
<tr>
<td>53666</td>
<td>15-721</td>
<td>C</td>
</tr>
</tbody>
</table>

### First SQL Example

**SELECT**

```
SELECT cid 
FROM enrolled 
WHERE grade = 'C'
```

Similar to...

$$\pi_{cid}(\sigma_{grade='C'}(enrolled))$$

But not quite….

<table>
<thead>
<tr>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>15-415</td>
<td>C</td>
</tr>
<tr>
<td>15-721</td>
<td>C</td>
</tr>
</tbody>
</table>

### Multi-Relation Queries

**SELECT**

```
SELECT name, cid 
FROM student, enrolled 
WHERE student.sid = enrolled.sid 
AND enrolled.grade = 'C'
```

Why preserve duplicates?

- Eliminating them is costly
- Users often don’t care.
Basic SQL Query Grammar

- **Relation-List**: A list of relation names
- **Target-List**: A list of attributes from the tables referenced in relation-list
- **Qualification**: Comparison of attributes or constants using operators \( =, \neq, <, >, \leq, \) and \( \geq \).

**SELECT Clause**

- Use `*` to get all attributes
  
  ```sql
  SELECT * FROM student
  ```

- Use `DISTINCT` to eliminate dupes
  
  ```sql
  SELECT DISTINCT cid FROM enrolled
  ```

- Target list can include expressions
  
  ```sql
  SELECT name, gpa*1.05 FROM student
  ```

**FROM Clause**

- Binds tuples to variable names
  
  ```sql
  SELECT * FROM student, enrolled
  WHERE student.sid = enrolled.sid
  ```

- Define what kind of join to use
  
  ```sql
  SELECT student.*, enrolled.grade
  FROM student LEFT OUTER JOIN enrolled
  WHERE student.sid = enrolled.sid
  ```

**WHERE Clause**

- Complex expressions using `AND`, `OR`, and `NOT`
  
  ```sql
  SELECT * FROM enrolled
  WHERE grade = 'C'
  AND (cid = '15-415' OR NOT cid = '15-826')
  ```

- Special operators `BETWEEN`, `IN`
  
  ```sql
  SELECT * FROM enrolled
  WHERE (sid BETWEEN 56000 AND 57000)
  AND cid IN ('15-415', '15-721')
  ```
The AS keyword can also be used to rename tables and columns in SELECT queries. Allows you to target a specific table instance when you reference the same table multiple times.

Renaming – Table Variables

- Get the name of the students that took 15-415 and got an ‘A’ or ‘B’ in the course.

```sql
SELECT student.name, enrolled.grade AS egrade
FROM student AS S, enrolled AS E
WHERE S.sid = E.sid
AND E.cid = '15-415'
AND E.grade IN ('A', 'B')
```

Renaming – Self-Join

- Find all unique students that have taken more than one course.

```sql
SELECT DISTINCT e1.sid
FROM enrolled AS e1, enrolled AS e2
WHERE e1.sid = e2.sid
AND e1.cid != e2.cid
```
More SQL

- **INSERT**
- **UPDATE**
- **DELETE**
- **TRUNCATE**

**INSERT**

- Provide target table, columns, and values for new tuples:

```sql
INSERT INTO student
    (sid, name, login, age, gpa)
VALUES
    (53888, 'Drake', 'drake@cs', 29, 3.5)
```

- Short-hand version:

```sql
INSERT INTO student VALUES
    (53888, 'Drake', 'drake@cs', 29, 3.5)
```

**UPDATE**

- **UPDATE** must list what columns to update and their new values (separated by commas).
- Can only update one table at a time.
- **WHERE** clause allows query to target multiple tuples at a time.

```sql
UPDATE student
    SET login = 'kwest@cs',
        age = age + 1
WHERE name = 'Kayne'
```

**DELETE**

- Similar to single-table **SELECT** statements.
- The **WHERE** clause specifies which tuples will deleted from the target table.
- The delete may cascade to children tables.

```sql
DELETE FROM enrolled
WHERE grade = 'F'
```
TRUNCATE

- Remove all tuples from a table.
- This is usually faster than **DELETE**, unless it needs to check foreign key constraints.

```
TRUNCATE student
```

Today's Party: OLTP

- Basic Queries
- Table Definition (DDL)
- NULLs
- String/Date/Time/Set/Bag Operations
- Output Redirection/Control

Table Definition (DDL)

```
CREATE TABLE <table-name>(
    [column-definition]*
    [constraints]*
) [table-options];
```

- **Column-Definition**: Comma separated list of column names with types.
- **Constraints**: Primary key, foreign key, and other meta-data attributes of columns.
- **Table-Options**: DBMS-specific options for the table (not **SQL-92**).

Table Definition Example

```
CREATE TABLE student (  
   sid   INT,  
   name  VARCHAR(16),  
   login VARCHAR(32),  
   age   SMALLINT,  
   gpa   FLOAT  
);  
CREATE TABLE enrolled (  
   sid   INT,  
   cid   VARCHAR(32),  
   grade CHAR(1)  
);  
```

Integer Range
Variable String Length
Fixed String Length
Common Data Types

- CHAR(n), VARCHAR(n)
- TINYINT, SMALLINT, INT, BIGINT
- NUMERIC(p, d), FLOAT, DOUBLE, REAL
- DATE, TIME
- BINARY(n), VARBINARY(n), BLOB

Useful Non-standard Types

- TEXT
- BOOLEAN
- ARRAY
- Geometric primitives
- XML/JSON
- Some systems also support user-defined types.

Integrity Constraints

```sql
CREATE TABLE student (  
   sid   INT PRIMARY KEY,  
   name  VARCHAR(16),  
   login VARCHAR(32) UNIQUE,  
   age   SMALLINT CHECK (age > 0),  
   gpa   FLOAT  
);  
CREATE TABLE enrolled (  
   sid   INT REFERENCES student (sid),  
   cid   VARCHAR(32) NOT NULL,  
   grade CHAR(1),  
   PRIMARY KEY (sid, cid)  
);  
```

Primary Keys

- Single-column primary key:

```sql
CREATE TABLE student (  
   sid   INT PRIMARY KEY,  
);  
```

- Multi-column primary key:

```sql
CREATE TABLE enrolled (  
   PRIMARY KEY (sid, cid)  
);  
```
Foreign Key References

- Single-column reference:

```
CREATE TABLE enrolled (
   sid  INT REFERENCES student (sid),
   ...
)
```

- Multi-column reference:

```
CREATE TABLE enrolled (
   ...
   FOREIGN KEY (sid, ...)
   REFERENCES student (sid, ...)
)
```

You can define what happens when the parent table is modified:
- CASCADE
- RESTRICT
- NO ACTION
- SET NULL
- SET DEFAULT

Delete/update the enrollment information when a student is changed:

```
CREATE TABLE enrolled (
   ...
   FOREIGN KEY (sid)
   REFERENCES student (sid)
   ON DELETE CASCADE
   ON UPDATE CASCADE
)
```

Value Constraints

- Ensure one-and-only-one value exists:

```
CREATE TABLE student (
   login VARCHAR(32) UNIQUE,
)
```

- Make sure a value is not null:

```
CREATE TABLE enrolled (
   cid   VARCHAR(32) NOT NULL,
)
```
Value Constraints

- Make sure that an expression evaluates to true for each row in the table:

```sql
CREATE TABLE student (  
    age   SMALLINT CHECK (age > 0),
)
```

- Can be expensive to evaluate, so tread lightly...

Auto-Generated Keys

- Automatically create a unique integer id for whenever a row is inserted (last + 1).

- Implementations vary wildly:
  - SQL:2003 → IDENTITY
  - MySQL → AUTO_INCREMENT
  - Postgres → SERIAL
  - SQL Server → SEQUENCE
  - DB2 → SEQUENCE
  - Oracle → SEQUENCE

Conditional Table Creation

- **IF NOT EXISTS** prevents the DBMS from trying to create a table twice.

```sql
CREATE TABLE IF NOT EXISTS student (  
    sid   INT PRIMARY KEY AUTO_INCREMENT,
    name  VARCHAR(16),
    login VARCHAR(32) UNIQUE,
    age   SMALLINT CHECK (age > 0),
    gpa   FLOAT
);  
```

Auto-Generated Keys

- Create a table `student` with an auto-incrementing primary key in MySQL:

```sql
CREATE TABLE student (  
    sid   INT PRIMARY KEY AUTO_INCREMENT,
    ...  
);
```

- Insert a row into the `student` table:

```sql
INSERT INTO student  
    (sid, name, login, age, gpa)  
VALUES  
    (NULL, "Drake", "drake@cs", 29, 4.0);
```
Dropping Tables

- Completely removes a table from the database. Deletes everything related to the table (e.g., indexes, views, triggers, etc):

```
DROP TABLE student;
```

- Can also use `IF EXISTS` to avoid errors:

```
DROP TABLE IF EXISTS student;
```

Modifying Tables

- SQL lets you add/drop columns in a table after it is created:

```
ALTER TABLE student
  ADD COLUMN phone VARCHAR(32) NOT NULL;

ALTER TABLE student
  DROP COLUMN login;
```

```
This is really expensive!!! Tread lightly...
```

Modifying Tables

- You can also modify existing columns (rename, change type, change defaults, etc):

```
ALTER TABLE student
  ALTER COLUMN login TYPE VARCHAR(32);  
```

```
ALTER TABLE student
  CHANGE COLUMN login login VARCHAR(32);
```

Accessing Table Schema

- You can query the DBMS’s internal `INFORMATION_SCHEMA` catalog to get info about the database.

- ANSI standard set of read-only views that provide info about all of the tables, views, columns, and procedures in a database.

- Every DBMS also have non-standard shortcuts to do this.
Accessing Table Schema

• List all of the tables in the current database:

SELECT * FROM INFORMATION_SCHEMA.TABLES WHERE table_catalog = '<db name>'

\d Postgres
SHOW TABLES; MySQL
.tables; SQLite

• List the column info for the student table:

SELECT * FROM INFORMATION_SCHEMA.COLUMNS WHERE table_name = 'student'

\d student Postgres
DESCRIBE student; MySQL
.schema student; SQLite

NULLs

• The “dirty little secret” of SQL, since it can be a value for any attribute.

<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>Kayne</td>
<td>kayne@cs</td>
<td>39</td>
<td>4.0</td>
</tr>
<tr>
<td>53688</td>
<td>Bieber</td>
<td>jbieber@cs</td>
<td>22</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>
<tr>
<td>53900</td>
<td>Andy</td>
<td>pavlo@cs</td>
<td>35</td>
<td>NULL</td>
</tr>
</tbody>
</table>

• What does this mean?
  – Is that we do not know Andy’s GPA?
  – Or does Andy not have a GPA?
NULLs

- Find all the students with a null GPA.

<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>Kayne</td>
<td>kayne@cs</td>
<td>39</td>
<td>4.0</td>
</tr>
<tr>
<td>53688</td>
<td>Bieber</td>
<td>jbieber@cs</td>
<td>22</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>
<tr>
<td>53900</td>
<td>Andy</td>
<td>pavlo@cs</td>
<td>35</td>
<td>NULL</td>
</tr>
</tbody>
</table>

SELECT * FROM student WHERE gpa = NULL

• Find all the students with a null GPA.

<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>Kayne</td>
<td>kayne@cs</td>
<td>39</td>
<td>4.0</td>
</tr>
<tr>
<td>53688</td>
<td>Bieber</td>
<td>jbieber@cs</td>
<td>22</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>
<tr>
<td>53900</td>
<td>Andy</td>
<td>pavlo@cs</td>
<td>35</td>
<td>NULL</td>
</tr>
</tbody>
</table>

• Arithmetic operations with NULL values is always NULL.

SELECT 1+NULL AS add_null,
       1-NULL AS sub_null,
       1*NULL AS mul_null,
       1/NULL AS div_null;

<table>
<thead>
<tr>
<th>add_null</th>
<th>sub_null</th>
<th>mul_null</th>
<th>div_null</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>

• Comparisons with NULL values varies.

SELECT true = NULL AS eq_bool,
       true != NULL AS neq_bool,
       true AND NULL AS and_bool,
       NULL = NULL AS eq_null,
       NULL IS NULL AS is_null;

<table>
<thead>
<tr>
<th>eq_bool</th>
<th>neq_bool</th>
<th>and_bool</th>
<th>eq_null</th>
<th>is_null</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>TRUE</td>
</tr>
</tbody>
</table>
Today's Class: OLTP

- Basic Queries
- Table Definition (DDL)
- NULLs
- String/Date/Time/Set/Bag Operations
- Output Redirection/Control

String Operations

- **LIKE** is used for string matching.
- String-matching operators
  - “%” Matches any substring (incl. empty).
  - “_” Matches any one character

```
SELECT * FROM enrolled AS e
WHERE e.cid LIKE '15-%'
```

```
SELECT * FROM student AS s
WHERE s.login LIKE '%@c_'
```

String Operations

- **SQL-92** defines string functions.
  - Many DBMSs also have their own unique functions
  - Can be used in either output and predicates:

```
SELECT SUBSTRING(name,0,5) AS abbrv_name
FROM student WHERE sid = 53688
```

```
SELECT * FROM student AS s
WHERE UPPER(e.name) LIKE 'KAY%'
```
**String Operations**

- SQL standard says to use `||` operator to concatenate two or more strings together.

```sql
SELECT name FROM student
WHERE login = LOWER(name) || '@cs'
```

**Date/Time Operations**

- Operations to manipulate and modify `DATE/TIME` attributes.
- Can be used in either output and predicates.
- *Support/syntax varies wildly…*

**Demo: Get the # of days since the beginning of the year.**

**Set/Bag Operations**

- Set Operations:
  - `UNION`
  - `INTERSECT`
  - `EXCEPT`

- Bag Operations:
  - `UNION ALL`
  - `INTERSECT ALL`
  - `EXCEPT ALL`

**Set Operations**

- Returns ids of students with or without an enrollment record.

```sql
(SELECT sid FROM student) UNION (SELECT sid FROM enrolled)
```

- Returns students with an enrollment record.

```sql
(SELECT sid FROM student) INTERSECT (SELECT sid FROM enrolled)
```

- Returns students without an enrollment record.

```sql
(SELECT sid FROM student) EXCEPT (SELECT sid FROM enrolled)
```
Today's Class: OLTP

- Basic Queries
- Table Definition (DDL)
- NULLs
- String/Date/Time/Set/Bag Operations
- Output Redirection/Control

Output Redirection

- Store query results in another table:
  - Table must not already be defined.
  - Table will have the same # of columns with the same types as the input.

```
CREATE TABLE CourseIds
(SELECT DISTINCT cid INTO CourseIds
  FROM enrolled);
```

MySQL

```
SELECT DISTINCT cid INTO CourseIds
  FROM enrolled;
```

- Insert tuples from query into another table:
  - Inner `SELECT` must generate the same columns as the target table.
  - DBMSs have different options/syntax on what to do with duplicates.

```
INSERT INTO CourseIds
(SELECT DISTINCT cid FROM enrolled);
```

Output Control

- `ORDER BY <column*> [ASC|DESC]`
  - Order the output tuples by the values in one or more of their columns.

```
SELECT sid, grade FROM enrolled
WHERE cid = '15-721'
ORDER BY grade
```

```
<table>
<thead>
<tr>
<th>sid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>53123</td>
<td>A</td>
</tr>
<tr>
<td>53334</td>
<td>A</td>
</tr>
<tr>
<td>53650</td>
<td>B</td>
</tr>
<tr>
<td>53666</td>
<td>D</td>
</tr>
</tbody>
</table>
```

```
SELECT sid FROM enrolled
WHERE cid = '15-721'
ORDER BY grade DESC, sid ASC
```

```
<table>
<thead>
<tr>
<th>sid</th>
</tr>
</thead>
<tbody>
<tr>
<td>53666</td>
</tr>
<tr>
<td>53650</td>
</tr>
<tr>
<td>53123</td>
</tr>
<tr>
<td>53334</td>
</tr>
</tbody>
</table>
```
Output Control

- **LIMIT <count> [offset]**
  - Limit the # of tuples returned in output.
  - Can set an offset to return a “range”

```
SELECT sid, name FROM student
WHERE login LIKE '%@cs'
LIMIT 10
```
**First 10 rows**

```
SELECT sid, name FROM student
WHERE login LIKE '%@cs'
LIMIT 20 OFFSET 10
```
**Skip first 10 rows, Return the following 20**

Additional Information

- Online SQL validators:
  - [http://format-sql.com](http://format-sql.com)

- When in doubt, try it out!

Next Class: OLAP

- Aggregations + Group By
- Complex Joins
- Views
- Nested Queries
- Common Table Expressions
- Database Application Example