CMU SCS

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

C. Faloutsos – A. Pavlo

Lecture#6: Fun with SQL (part2)

CMU SCS

Today's Party

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

CMU SCS

Example Database

STUDENT

sid	name	login	age	gpa
53666	Faloutsos	christos@cs	45	4.0
53688	Bieber	jbieber@cs	21	3.9
53677	Tupac	shakur@cs	26	3.5

ENROLLED

sid	cid	grade
53831	Pilates101	C
53688	Reggae203	D
53688	Topology112	A
53666	Massage105	D

CMU SCS

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**).

6

CMU SCS

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

CMU SCS

Common Data Types

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

CMU SCS

Comment About BLOBs

- Don't store large files in your database!
- Put the file on the filesystem and store a URI in the database.
- Many app frameworks will do this automatically for you.
- More information:
  - *To BLOB or Not To BLOB: Large Object Storage in a Database or a Filesystem?*

CMU SCS

Useful Non-standard Types

- **TEXT**
- **BOOLEAN**
- **ARRAY**
- Some systems also support user-defined types.

CMU SCS

Integrity Constraints

```
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)
);
```

PKey Definition

Type Attributes

FKey Definition

11

CMU SCS

Primary Keys

- Single-column primary key:

```
CREATE TABLE student (
   sid   INT PRIMARY KEY,
   ⋮
```

- Multi-column primary key:

```
CREATE TABLE student (
   ⋮
   PRIMARY KEY (sid, name)
```


CMU SCS

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, …)
```


CMU SCS

Foreign Key References

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

CMU SCS

Foreign Key References

- 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
```


CMU SCS

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,
```


CMU SCS

Value Constraints

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

```
CREATE TABLE enrolled (
  age   SMALLINT CHECK (age > 0),
```

- *Can be expensive to evaluate, so tread lightly…*

CMU SCS

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`**

CMU SCS

Auto-Generated Keys

MySQL

```
CREATE TABLE student (
   sid   INT PRIMARY KEY AUTO_INCREMENT,
   ⋮
```

```
INSERT INTO student
  (sid, name, login, age, gpa)
VALUES
  (NULL, "Christos", "@cs", 45, 4.0);
```


CMU SCS

Conditional Table Creation

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

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

CMU SCS

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;
```


CMU SCS

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…*

CMU SCS

Modifying Tables

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

Postgres

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

MySQL

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

CMU SCS

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.

CMU SCS

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
```


CMU SCS

Accessing Table Schema

- 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
```

CMU SCS

Today's Party

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

CMU SCS

Example Database

STUDENT

sid	name	login	age	gpa
53666	Faloutsos	christos@cs	45	4.0
53688	Bieber	jbieber@cs	21	3.9
53655	Tupac	shakur@cs	26	3.5

ENROLLED

sid	cid	grade
53666	Pilates101	C
53688	Reggae203	D
53688	Topology112	A
53666	Massage105	D

COURSE

cid	name
Pilates101	Pilates
Reggae203	20th Century Reggae
Topology112	Topology + Squirrels
Massage105	Massage & Holistic Therapy

CMU SCS

Join Query Grammar

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

- **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 =, ≠, <, >, ≤, and ≥.

CMU SCS

INNER JOIN

sid	name	login	age	gpa
53666	Faloutsos	christos@cs	45	4.0
53688	Bieber	jbieber@cs	21	3.9
53655	Tupac	shakur@cs	26	3.5

sid	cid	grade
53666	Pilates101	C
53688	Reggae203	D
53688	Topology112	A
53666	Massage105	D

```
SELECT name, cid, grade
  FROM student INNER JOIN enrolled
    ON student.sid = enrolled.sid
```

name	cid	Grade
Bieber	Reggae203	D
Bieber	Topology112	A
Faloutsos	Massage105	D
Faloutsos	Pilates101	C

30

CMU SCS

INNER JOIN

- Short-hand version

```
SELECT student.sid, cid, grade
  FROM student, enrolled
 WHERE student.sid = enrolled.sid
```

Faloutsos/Pavlo CMU SCS 15-415/615 31

CMU SCS

OUTER JOIN

sid	name	login	age	gpa
53666	Faloutsos	christos@cs	45	4.0
53688	Bieber	jbieber@cs	21	3.9
53677	Tupac	shakur@cs	26	3.5

sid	cid	grade
53666	Pilates101	C
53688	Reggae203	D
53688	Topology112	A
53666	Massage105	D

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

name	cid	Grade
Bieber	Reggae203	D
Bieber	Topology112	A
Faloutsos	Massage105	D
Faloutsos	Pilates101	C
Tupac	NULL	NULL

32

CMU SCS

OUTER JOIN

sid	name	login	age	gpa
53666	Faloutsos	christos@cs	45	4.0
53688	Bieber	jbieber@cs	21	3.9
53677	Tupac	shakur@cs	26	3.5

sid	cid	grade
53666	Pilates101	C
53688	Reggae203	D
53688	Topology112	A
53666	Massage105	D

```
SELECT student.sid, cid, grade
  FROM enrolled RIGHT OUTER JOIN student
    ON student.sid = enrolled.sid
```

name	cid	Grade
Bieber	Reggae203	D
Bieber	Topology112	A
Faloutsos	Massage105	D
Faloutsos	Pilates101	C
Tupac	**NULL**	**NULL**

33

CMU SCS

Join Types

```
SELECT * FROM A JOIN B ON A.id = B.id
```

Join Type	Description
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

CMU SCS

Today's Party

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

CMU SCS

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!*

CMU SCS

View Example

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

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

Original Table

sid	name	login	age	gpa
53666	Faloutsos	christos@cs	45	4.0
53688	Bieber	jbieber@cs	21	3.9

View

sid	name	login
53666	Faloutsos	christos@cs
53688	Bieber	jbieber@cs

CMU SCS

View Example

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

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

cid	avg_age
Massage105	45.0
Pilates101	45.0
Topology112	21.0
Reggae203	21.0

CMU SCS

Views vs. SELECT INTO

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

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

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

39

CMU SCS

Today's Party

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

CMU SCS

Nested Queries

- Queries containing other queries

"outer query" *"inner query"*

- Inner query:
  - Can appear in **FROM** or **WHERE** clause

```
SELECT cname FROM borrower WHERE
cname IN (SELECT cname FROM depositor)
```

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

cname
Johnson
Smith
Jones
Smith

CMU SCS

Nested Queries

- Find the names of students in 'Massage105'

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

"sid in the set of people that take Massage105"

CMU SCS

Nested Queries

- Find the names of students in 'Massage105'

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


CMU SCS

Nested Queries

- Find the names of students in 'Massage105'

```
SELECT name FROM student
 WHERE sid IN (
   SELECT sid FROM enrolled
    WHERE cid = 'Massage105'
 )
```

name
Faloutsos

CMU SCS

Nested Queries

- **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.*

CMU SCS

Nested Queries

- Find the names of students in 'Massage105'

```
SELECT name FROM student
 WHERE sid = ANY(
   SELECT sid FROM enrolled
    WHERE cid = 'Massage105'
 )
```

name
Faloutsos

CMU SCS

Nested Queries

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

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

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

sid	name
53688	Tupac

CMU SCS

Nested Queries

- Find student record with the highest id.

```
SELECT sid, name FROM student
 WHERE ...
```

"is greater than every other sid"

CMU SCS

Nested Queries

- Find student record with the highest id.

```
SELECT sid, name FROM student
 WHERE sid  is greater than every
   SELECT sid FROM enrolled
```


CMU SCS

Nested Queries

- Find student record with the highest id.

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

sid	name
53688	Bieber

CMU SCS

Nested Queries

- Find student record with the highest id.

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

sid	name
53688	Bieber

CMU SCS

Nested Queries

- Find all courses that nobody is enrolled in.

```
SELECT * FROM course
 WHERE ...
```

"with no tuples in the 'enrolled' table"

cid	name
Pilates101	Pilates
Reggae203	20th Century Reggae
Karate101	Karate Kid Aerobics
Topology112	Topology + Squirrels
Massage105	Massage & Holistic Therapy

sid	cid	grade
53666	Pilates101	C
53688	Reggae203	D
53688	Topology112	A
53666	Massage105	D

CMU SCS

Nested Queries

- Find all courses that nobody is enrolled in.

```
SELECT * FROM course
 WHERE NOT EXISTS(
   tuples in the 'enrolled' table
)
```

CMU SCS

Nested Queries

- Find all courses that nobody is enrolled in.

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


CMU SCS

Today's Party

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

CMU SCS

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**

CMU SCS

Trigger Example

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

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


CMU SCS

Trigger Example

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

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

Postgres

CMU SCS

Trigger Example

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

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

Postgres

CMU SCS

Today's Party

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

CMU SCS

Outline of an DB application

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

CMU SCS

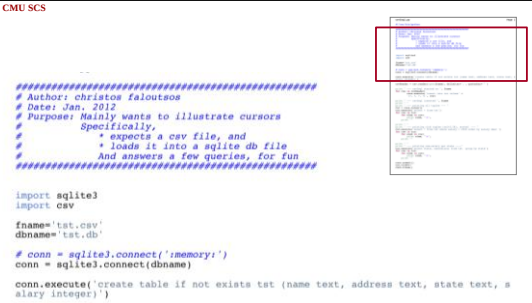
```
#!/usr/bin/python
##########################################
# Author: christos faloutsos
# Date: Jan. 2012
# Purpose: Mainly wants to illustrate cursors
#          Specifically,
#            * expects a csv file, and
#            * loads it into a sqlite db file
#            * And answers a few queries, for fun
##########################################

import sqlite3
import csv

fname='tst.csv'
dbname='tst.db'

# conn = sqlite3.connect(':memory:')
conn = sqlite3.connect(dbname)

conn.execute('create table if not exists tst (name text, address text, state text, s
alary integer)')

csvReader = csv.reader(open(fname), delimiter=',', quotechar='"')

print ' --- csv2sql started on ', fname
for row in csvReader:
    conn.execute('insert into tst values' +
      '(?, ?, ?, ?)', row)

print ' --- csv2sql inserted ', fname
print ' '
print ' --- printing all tuples --- '
cur = conn.cursor()
cur.execute('select * from tst')
for row in cur:
    for elem in row:
        print elem, '\t',
    print ''

print ' '
print ' --- printing rich tuples (sal>2.5K), sorted --- '
cur.execute('select * from tst where salary > 2500 order by salary desc ')
for row in cur:
    for elem in row:
        print elem, '\t',
    print ''

print ' '
print ' --- printing sum-salary per state ----'
cur.execute('select state, sum(salary) from tst  group by state')
for row in cur:
    for elem in row:
        print elem, '\t',
    print ''

conn.commit()
cur.close()
conn.close()
```

CMU SCS

Sample Python Code

- http://www.cs.cmu.edu/~christos/courses/dbms.S14/PYTHON-examples/csv2sql.py
- Or follow instructions at
- http://www.cs.cmu.edu/~christos/courses/dbms.S14/PYTHON-examples/

CMU SCS



```
##############################################
# Author: christos faloutsos
# Date: Jan, 2012
# Purpose: Mainly wants to illustrate cursors
#          Specifically,
#             * expects a csv file, and
#             * loads it into a sqlite db file
#             And answers a few queries, for fun
##############################################

import sqlite3
import csv

fname='tst.csv'
dbname='tst.db'

# conn = sqlite3.connect(':memory:')
conn = sqlite3.connect(dbname)

conn.execute('create table if not exists tst (name text, address text, state text, s
alary integer)')
```


CMU SCS

```
print " --- csv2sql inserted ", fname
print " "
print " --- printing all tuples --- "
cur = conn.cursor()
cur.execute('select * from tst')
for row in cur:
    for elem in row:
        print elem, "\t",
    print ""
```

CMU SCS

Cursors

- Used to iterate through the results of query.
- Enables result rows to be processed sequentially.
  - *Runs counter to SQL's set-based nature.*
  - *Very inefficient in most cases!*

CMU SCS

```
print " --- csv2sql inserted ", fname
print " "
print " --- printing all tuples --- "
cur = conn.cursor()
cur.execute('select * from tst')
for row in cur:
    for elem in row:
        print elem, "\t",
    print ""
```


CMU SCS

```
print " --- csv2sql inserted ", fname
print " "
print " --- printing all tuples --- "
cur = conn.cursor()
cur.execute('select * from tst')
for row in cur:
    for elem in row:
        print elem, "\t",
    print ""
```

CMU SCS

```
print " "
print " --- printing sum-salary per state ----"
cur.execute('select state, sum(salary) from tst  group
for row in cur:
    for elem in row:
        print elem, "\t",
    print ""

conn.commit()
cur.close()
conn.close()
```


CMU SCS

Summary

Outline of an SQL application:

- Establish connection with db server
- Authenticate (user/password)
- Execute SQL statement(s) (using **cursors**)
- Process results
- Close connection

CMU SCS

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.

CMU SCS

ORM Example

```
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)
```

72

CMU SCS

ORM Example

```
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),
);
```

CMU SCS

ORM Libraries

- Standalone:
  - Hibernate (Java)
  - SQLAlchemy (Python)
  - Doctrine (PHP)
- Integrated:
  - Django (Python)
  - ActiveRecord (Ruby on Rails)
  - CakePHP (PHP)