CS601: SQL Outer Join. JDBC

Olga Karpenko

The examples in this presentation are from the book "MySQL: Crash course" by Ben Forta

Announcements

- Lab 6 due tonight!
- Exam has been graded
  - Average 19.886
  - Graded out of 24
  - If got less than B-, come to office hours
- No quiz on Wednesday
- Project will come out tomorrow

Natural Join

- Join based on columns with the same name
- Eliminates multiple occurrences of the same column: Only one of each column is returned
  - SELECT *

    FROM customers NATURAL JOIN orders;

  - No need to specify what to join on
  - The cust_id column only appears once in the result

Natural Join

- Compare the results of the two queries:

```
SELECT *
FROM customers INNER JOIN orders
WHERE customers.cust_id=orders.cust_id;
```

and

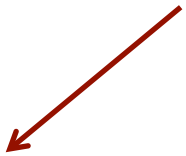
```
SELECT *
FROM customers Natural JOIN orders;
```

Natural Join

- Compare the results of the two queries:

```
SELECT *
FROM customers INNER JOIN orders
WHERE customers.cust_id=orders.cust_id;
```

and



cust_id column will be included only once

```
SELECT *
FROM customers Natural JOIN orders;
```

Outer Joins: Motivation

- If we want to include rows from one table that have no related rows in another table
- Examples:
  - Count how many orders each customer placed, including customers who haven't placed an order yet
  - List all products with order quantities including products not ordered by anyone

Outer Join: Example

- A list of all customers and their orders
- Include customers who have no orders

```
SELECT customers.cust_id, orders.order_num
FROM customers LEFT OUTER JOIN orders
ON customers.cust_id=orders.cust_id;
```

Left Outer Join

- A left outer join retains all of the rows of the “left” table
  - regardless of whether there is a row that matches on the “right” table.
  - The “left” table is the table that comes first in the join statement

Outer Join

- List all products with order quantities including products not ordered by anyone

```
SELECT products.prod_id, quantity
FROM products LEFT OUTER JOIN orderitems
ON orderitems.prod_id =products.prod_id;
```

Right Outer Join

- A right outer join retains all of the rows of the “right” table
  - regardless of whether there is a row that matches on the “left” table
  - The “right” table is the table that comes last in the join statement

JDBC

JDBC

- Java Database Connectivity
- Java API for connecting to relational databases
- java.sql and javax.sql packages
- Requires a driver for specific DBMS

Driver

- MySql Connector:
  - https://dev.mysql.com/downloads/connector/j/
- Provided to you in Examples/lib
  - mysql-connector-java-5.1.44-bin.jar

Architecture

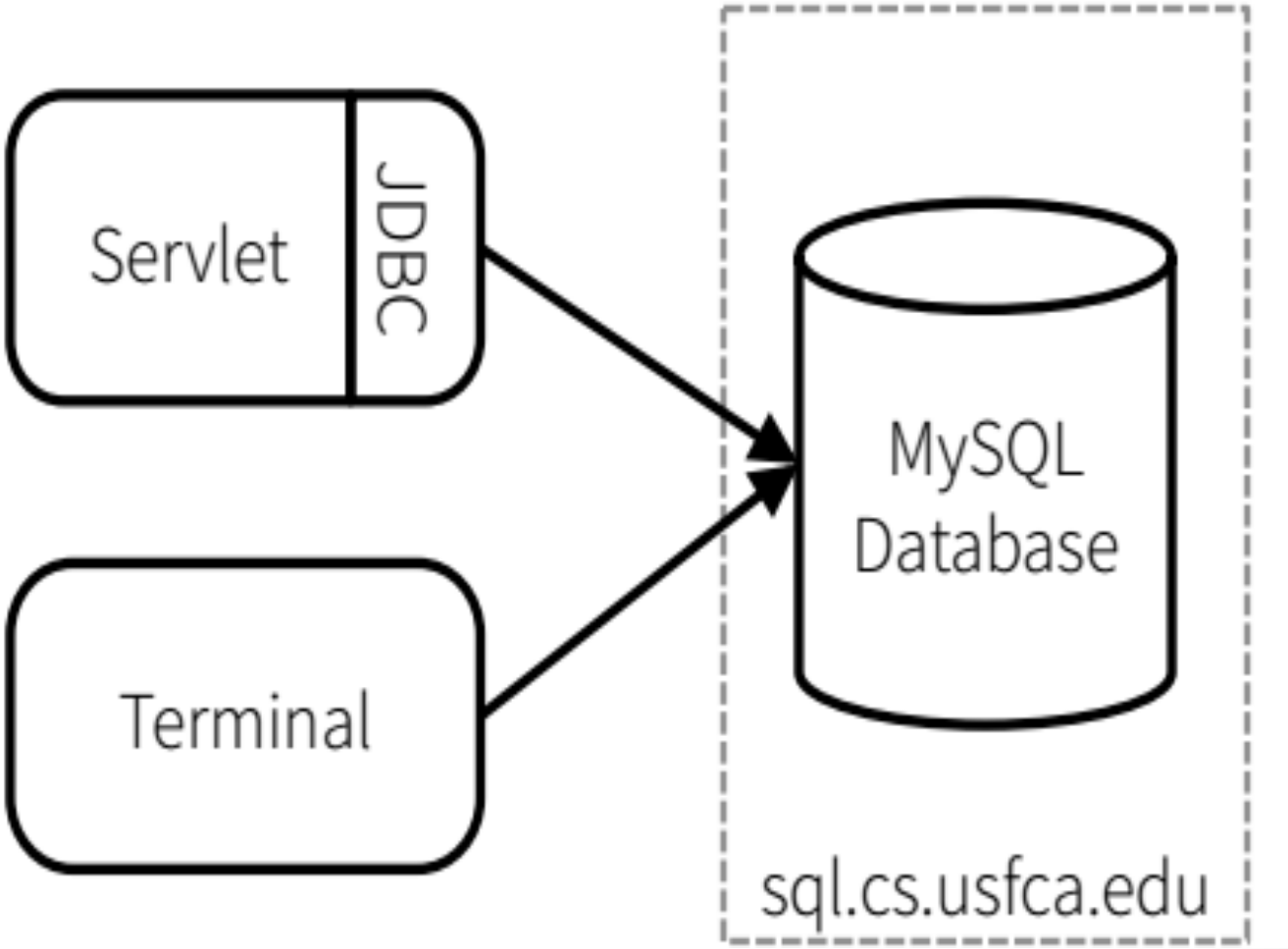


Diagram is courtesy of Prof. Engle

Accessing from Campus

- Connect to sql.cs.usfca.edu
- If using JDBC
  - Update the provided database.properties file

Accessing Remotely

- Must create ssh tunnel

ssh username@stargate.cs.usfca.edu

-L 3307:sql.cs.usfca.edu:3306 –N

- Point database location to the local port on the tunnel:

jdbc:mysql://localhost:3307/username

Examples

- SimpleJDBCExample

SQL Injections

- User Input is specially crafted so that when it's inserted into SQL query, it's treated as SQL code
- Possible Goals
  - Determine schema
  - Extract data
  - Gain unauthorized access to parts of the database
  - Add/modify/delete data

http://www.slideshare.net/pallavibiswas1/sql-injection-16984356

Example

Username 'okarpenko'

Password 'usf#$%'

```
SELECT  *
FROM login_users
WHERE user='okarpenko' AND
password='usf#$%';
```

http://www.slideshare.net/pallavibiswas1/sql-injection-16984356

SQL Injection

Username 'okarpenko' OR 0=0; /*

Password '*/---'

```
SELECT *
FROM login_users
WHERE user='okarpenko' OR 0=0; /* AND
password=' */---'
```

Able to pull information of this user without providing the password!

http://www.slideshare.net/pallavibiswas1/sql-injection-16984356

Different Types of Attacks

- Enter malicious string -> code will execute immediately
- The attacker injects into DB table. An attack is executed by another request

- Cause: Poor input validation

http://www.slideshare.net/pallavibiswas1/sql-injection-16984356

Prepared Statements

- A template for sql statement

```
PreparedStatement statement =
connection.prepareStatement("
SELECT username
FROM login_users
WHERE username = ?");
```

- Precompiled -> More efficient
- User entered values are inserted as literal values, not executable SQL code

Prepared Statements

```
PreparedStatement statement =
connection.prepareStatement("
INSERT INTO login_users
(username, password, usersalt)
VALUES (?, ?, ?);");
```

See the Difference

```
1. Statement stmt = conn.createStatement("
INSERT INTO students
VALUES('" + user + "')");
stmt.execute();

vs
// Prepared Statement
2. Statement stmt = conn.prepareStatement("
INSERT INTO student
VALUES(?)");
stmt.setString(1, user);
stmt.execute();
```

http://stackoverflow.com/questions/1582161/how-does-a-preparedstatement-avoid-or-prevent-sql-injection

See the Difference

If User Input is:

```
Robert'); DROP TABLE students; --
```

1. Table `students` is deleted!

vs

2. Insert the following user into the table:

"Robert'); DROP TABLE students;--"

http://stackoverflow.com/questions/1582161/how-does-a-preparedstatement-avoid-or-prevent-sql-injection