CS320 Web and Internet Programming

Database Access

Chengyu Sun
California State University, Los Angeles

Web and Databases

- E-commerce sites
  - Products, order, customers
- News sites
  - Subscribers, articles
- Web boards
  - Users, postings
- … anywhere where a large amount of information needs to be managed safely and efficiently

Database vs. File

- More efficient search
- ACID
  - Automicity
  - Consistency
  - Isolation
  - Durability

Relational Model

- Proposed by Edgar F. Codd in earlier 1970's
- All major databases are relational

A Relational DB Example

Orders

OID	CID	ODATE	SDATE
001	001	4/29/2004	NULL
002	002	3/20/2004	3/37/2004

Products

PID	Description	Price
CPU01	Intel P4	$200
CPU02	Intel P3	$49
CPU03	AthlonXP	$100
MBD01	ASUS	$128
MBD02	TYAN	$400

Customers

CID	FNAME	LNAME	ADDRESS
001	Chengyu	Sun	Street #215
002	Steve	Sun	Street #711

Order Details

OID	PID	Quantity
001	CPU01	2
001	MBD02	2
002	CPU02	1

Terminology

- Database
- Table, relation
- Attribute, field
  - Type
- Record, tuple, row
- Column

SQL

- Standard query language of relational databases
- Supported by all major relational databases with some variations
- Pronunciation

MySQL

- A popular Open Source DBMS
- Super fast SELECT
- Non ACID-compliant until very recent
- Why MySQL grew so fast?
  - http://www.oreillynet.com/pub/wlg/4715

MySQL on the CS Server

- Version 3.23
- One database per user
  - DB name is the same as the server account user name. E.g. `studentxx`
- Connect to the database
  - `mysql –u csun –p`
  - Username and password are the same as the ones for the server account

Some MySQL Commands …

- Help
  - \h or help;
- Quite MySQL client
  - \q or quit;
- Change password
  - set password = password ('something');
- Show databases
  - show databases;

… Some MySQL Commands

- Use database
  - use csun;
- Show tables
  - show tables;
- Show table schema
  - describe Products;
- Run a script
  - \. demo.sql or source demo.sql;

Create a Table

```
create table table_name (
        field_name field_type [NOT NULL] [DEFAULT value],
        field_name field_type [NOT NULL] [DEFAULT value],
        …
        [PRIMARY KEY(field_name, …)]
);
```

```
create table Products (
      prod_id char(8) not null,    -- product id
      description text,            -- product description
      price decimal(12,2),         -- price
      primary key (prod_id)
);
```

Field Types

- Numerical types – *(M,D) [UNSIGNED]*
  - Int(M)
  - Float(M,D), Double(M,D)
  - Decimal(M,D)
- String types – (M)
  - Char(M), Varchar(M)
  - Text
- Date and time
  - DATE – 'YYYY-MM-DD'
  - TIME – 'HH-MM-SS'

Populate Tables

- Insert a record
  - insert into Orders values (1000, 1, '2004-04-29', '2004-05-01');
  - insert into Orders values (1001, 2, '2004-05-01', NULL);
- Load a data file
  - load data local infile 'orders.txt' into table Orders;
- Import a data file (at command prompt)
  - mysqlimport –u csun –p csun Orders.txt
    - \N for NULL

Search for Records

select field(s) from table(s) where condition(s);

- select description, price from Products;
- select * from Products;
- select * from Products where price < 300;
- select * from Products where prod_id = 'cpu-0001';

Pattern Matching

- LIKE, REGEXP
  - % -- any zero or more characters
  - . – any single charater
  - [abc], [a-z], [0-9] – range
  - * -- zero or more instances of the preceding character
  - ^ -- beginning of a string
  - $ -- end of a string
- select * from Products where description like '%intel%';

Update Records

update table set field=value [, …] where condition(s);

- update Products set price=320 where prod_id = 'cpu-0001';
- update Products set price=200, description='Intel Pentium M 1.7GHz' where prod_id = 'cpu-0001';

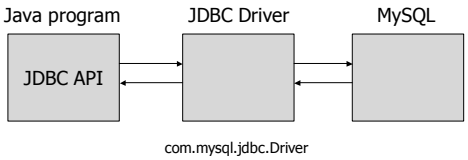
Delete Data

delete from table where condition(s);

- delete from Orders;
- delete from Orders where order_date < '2003-12-31' and ship_date is not null;
- Drop a database
  - drop database csun; -- Don't do this!
- Drop a table
  - drop table Products;

JDBC

- An interface between Java programs and SQL databases



JDBC Basics …

- import java.sql.*;
- Load driver
  - Class.forName("com.mysql.jdbc.Driver")
- Create connection
  - Connection c = DriverManager.getConnection(URL);
  - URL
    - jdbc:mysql://[hostname]/[dbname][?user=csun&password=something]

… JDBC Basics

- Create statement
  - Statement stmt = c.createStatement();
    - stmt.executeQuery()
    - stmt.executeUpdate()
- Get result back
  - ResultSet rs

http://java.sun.com/j2se/1.3/docs/guide/jdbc/

DB Query Results

- In a program, we want to
  - Access each row
  - Access column in a row
  - Access column names

select * from UserInfo;

user	course
csun	cs122
csun	cs201
csun	cs320

JDBC ResultSet – Row Access

- `next()` – move cursor down one row
  - `true` if the current row is valid
  - `false` if no more rows
  - Cursor starts from *before the 1st row*

JDBC ResultSet – Column Access

- Access the columns of *current row*
- getXxx(String columnName)
  - E.g. getString("user");
- getXxx(int columnIndex)
  - columnIndex starts from 1
  - E.g. getString(1);

JDBC ResultSet – Access Column Names

```
ResultSetMetaData meta = rs.getMetaData();
```

- ResultSetMetaData
  - getColumnName(columnIndex)
    - Column name
  - getColumnLabel(columnIndex)
    - Column title for display or printout

Authentication Type of Queries

- For example, check whether a <username, password> pair is in database
- Only the size of the result set matters
  - 1 – authentication succeeded
  - 0 – authentication failed

JDBC ResultSet – Size

- No size() method?
- Something about *FetchSize*
  - getFetchSize()
  - setFetchSize(int nrows)

About Lab 2

- Implement the search in Java code
- Use the search capability of the database
  - Retrieve matching row(s)
  - Retrieve number of matching row(s)

JSTL SQL

- sql:transaction
- sql:query
- sql:update
- sql:param
- sql:dateParam
- sql:setDataSource

sql:setDataSource

- `var` – data source name. Only needed when you have multiple db sources.
- `scope` – scope of the data source
- `driver` – "com.mysql.jdbc.Driver"
- `url` – "jdbc:mysql:///dbname"
- `user`
- `password`
- `dataSource`

sql:query

- `var` – name of the result set
- `scope` – scope of the result set
- `sql` – query statement
- `dateSource` – name of the data source
- `startRow`
- `maxRows` – max number of rows in the result set

sql:query Result Set

- javax.servlet.jsp.jstl.sql.Result
  - SortedMap[] getRows()
  - Object[][] getRowsByIndex()
  - String[] getColumnNames()
  - int getRowCount()
  - boolean isLimitedByMaxRows()

sql:query example 1

```
<sql:query var="prod" sql="SELECT * FROM Products"/>

<table>
  <c:forEach items="${prod.rows}" var="row">
    <c:forEach items="${row}" var="col">
      <tr>
        <td>${col.key}</td><td>${col.value}</td>
      </tr>
    </c:forEach>
  </c:forEach>
</table>
```

sql:query example 2

```
<sql:query var="prod">
  SELECT * FROM Products WHERE
       description LIKE '%Intel%' AND price < 300
</sql:query>

<table>
  <c:forEach items="${prod.rowsByIndex}" var="row">
  <tr>
    <c:forEach items="${row}" var="col">
        <td>${col}</td>
    </c:forEach>
  </tr>
  </c:forEach>
</table>
```

sql:query example 3

- Place holder and `<sql:param>`

```
<sql:query var="prod">

   SELECT * FROM Products WHERE
       description LIKE ? AND price < ?

  <sql:param value="%Intel%"/>
  <sql:param value="250"/>

</sql:query>
```

sql:update

- `var` – name of the result variable. int
  - number of rows affected by the update
  - 0 if the update statement doesn't return anything
- `scope`
- `sql`
- `dateSource` – name of the data source

sql:update example

```
<c:if test="${! empty param.price}">
    <sql:update var="r">
        UPDATE Products SET price = ? WHERE prod_id = 'CPU-0003'
        <sql:param value="${param.price}"/>
    </sql:update>
</c:if>
```