Web Development using Java, JSP, and Web Services

Relational Databases & JDBC

Lecture #7 2008

Terminology

- Database server software referred to as Database Management Systems (DBMS)
- Database schemas describe database structure
- Data ordered in tables, rows and columns
- Tables (relations) provide groupings of data
- Rows (tuples) contain data records
- Columns (attributes) provide field structure

Structure

- Keys are used for ids and references
- Data stored in rows (aka records, tuples)
- Data accessed and manipulated via SQL, JDBC etc
- Databases can store vast amounts of data
- Databases also contains simple functions
- Creation of custom functions (stored procedures) supported by most major database systems

Motivation

- Allows data storage to be structured after data access patterns
- Data can be indexed and hashed in memory - quick access
- A simple and expressive model that supports logic reasoning (first-order predicate logic)
- Simple and efficient data storage
- Database models support role-based development

Database Table

id	username	birthdate	email
1	test1	1970-01-01	test1@jsp.com
2	test2	1970-01-02	test2@jsp.com
3	test3	1970-01-03	test3@jsp.com
4	test4	1970-01-04	test4@jsp.com
5	test5	1970-01-05	test5@jsp.com

Transactions

- A way to provide safe concurrent access to databases
- Allows for several operations to be viewed as a single, atomic operation
- If an error occurs, all changes are undone (rollback)

Structured Query Language (SQL)

- Text-based language used to access databases
- Standardized versions exists (ANSI SQL)
- Most database vendors provide their own dialects
- Select queries are used to ask questions about data
- Insert, update and delete statements are used to manipulate databases
- Create, alter & drop statements are used to create, modify and destroy databases
- Most databases provide functions which can be called using SQL

SQL Queries

```
SELECT *
FROM profile
WHERE profile.id = '123'

SELECT username, birthdate, email
FROM profile
WHERE profile.id = '123'
```

SQL Queries

```
SELECT COUNT(*)
FROM profile
WHERE LEN(profile.email) > 0

SELECT *
FROM user, profile
WHERE user.id = profile.id
AND LEN(profile.email) > 0
ORDER BY profile.username
LIMIT 25                              <- MySQL dialect
```

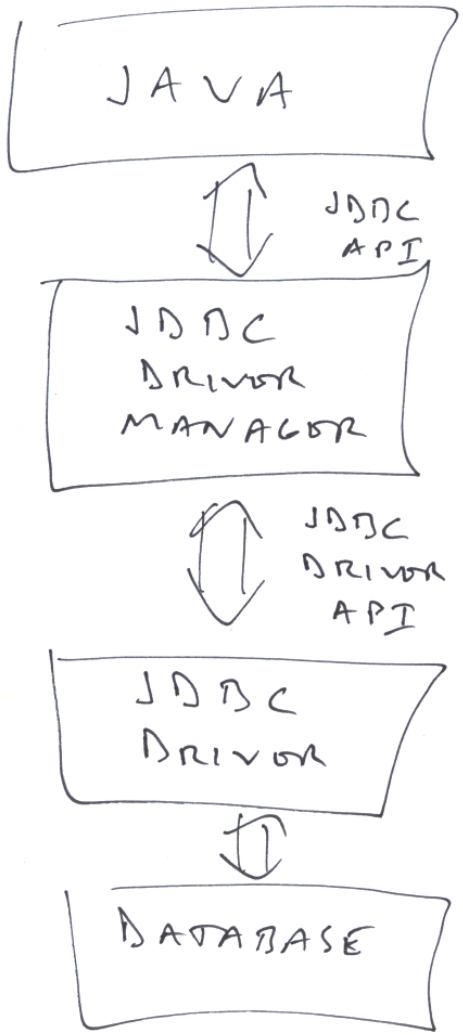
SQL Updates

```
INSERT INTO profile (id, username, birthdate, email)
VALUES (123,'test1','1970-01-01','test1@jsp.com')

UPDATE profile SET email = 'test1@jsp.com'
WHERE id = '123'

DELETE FROM profile
WHERE id = '123'
```

Java Database Connectivity (JDBC)

- Java API for database access
- Constructed around a JDBC Driver Manager
- Database vendors provide JDBC drivers
- Aims to virtualize Java database access
- Switch database without changes in Java code

JDBC


JAVA
JDBC API
JDBC DRIVER MANAGER
JDBC DRIVER API
JDBC DRIVER
DATABASE

JDBC Data Types

JDBC type	Java type
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
FLOAT, DOUBLE	double
BINARY, VARBINARY, LONGVARBINARY	byte[]
CHAR, VARCHAR, LONGVARCHAR	String
NUMERIC, DECIMAL	BigDecimal

JDBC Data Types

JDBC type	Java type
DATE	java.sql.Date
TIME, TIMESTAMP	java.sql.Timestamp
CLOB	Clob
BLOB	Blob
ARRAY	Array
DISTINCT	mapping underlying type
STRUCT	Struct
REF	Ref
JAVA_OBJECT	underlying Java class

Using JDBC

1. Load driver
2. Create database URL
3. Connect to database
4. Create a statement
5. Execute a query / update
6. Process results
7. Close connection

JDBC Drivers

- Data buffering
- Connection pooling
- Links to stored procedures
- Driver instantiated via dynamic class loading (required, need only be done once)

JDBC Driver Example

```
public static void instantiateDriver (String driver)
  throws ClassNotFoundException
{
  try
  {
    Class.forName(driver).newInstance();
  }
  catch (InstantiationException e)
  {
    throw new ClassNotFoundException(e.getMessage());
  }
  catch (IllegalAccessException e)
  {
    throw new ClassNotFoundException(e.getMessage());
  }
}
```

JDBC Driver Example

```
static
{
  // Apache Derby
  instantiateDriver("org.apache.derby.jdbc.EmbeddedDriver");

  // MySQL
  instantiateDriver("com.mysql.jdbc.Driver");

  // PostgreSQL
  instantiateDriver("org.postgresql.Driver");
}
```

JDBC Database URLs

jdbc:*dbms*:[//*host*:*port*/]*database*

- Provides JDBC database connection information
- Contains host, port, database (optionally username/password)

JDBC Database URL Examples

DBMS	URL
Apache Derby	jdbc:derby:*database*
MySQL	jdbc:mysql://*host*:3306/*database*
PostgreSQL	jdbc:postgresql://*host*:5432/*database*

JDBC Database Connections

- Represents a connection to a database
- May result in an underlying TCP/IP connection
- Connection may be compressed / encrypted
- Costly to create, reuse is recommended

JDBC Statement Types

- Statement - simple SQL statement
- PreparedStatement - precompiled SQL statement
- CallableStatement - database stored procedure

JDBC Statements

- SQL is constructed and parsed for each request
- Quick and easy
- Should be replace with prepared statements (when optimizing)

JDBC PreparedStatements

- Used to precompile a JDBC statement
- Parameters are added to the statement
- More efficient when queries are repeated

JDBC CallableStatements

- Used to call stored procedures in a database
- Parameters are added to call
- Stored procedure executes in the DBMS
- More efficient than prepared statements
- Requires a stored procedure
- Ties the application to a particular DBMS

JDBC ResultSets

- Interfaces to statement results
- Can be thought of as table views
- Doesn't actually contain all data returned from queries (requests data as needed)
- Can be used to both read and write data
- Exact behavior of ResultSets depends on statement type

JDBC Select

- Retrieves data from the database

```
Connection connection =
  DriverManager.getConnection(url,username,password)
Statement statement =
  connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                             ResultSet.CONCUR_READ_ONLY);
String sql = "SELECT * FROM profile";
ResultSet resultSet = statement.executeQuery(sql);
resultSet.next();
String id = resultSet.getString("id");
String username = resultSet.getString("username");
String birthdate = resultSet.getString("birthdate");
String email = resultSet.getString("email");
```

JDBC SQL Insert

- Inserts data into the database
- Data must be manually filtered / escaped

```
Connection connection =
  DriverManager.getConnection(url,username,password)
Statement statement =
  connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                             ResultSet.CONCUR_UPDATABLE);
String sql =
  "INSERT INTO profile (id, username, birthdate, email) " +
  "VALUES (123,'test1','1970-01-01','test1@jsp.com')";
ResultSet resultSet = statement.execute(sql);
```

JDBC ResultSet Insert

- Inserts data into the database
- Data is filtered / escaped by JDBC

```
Connection connection =
  DriverManager.getConnection(url,username,password)
Statement statement =
  connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                             ResultSet.CONCUR_UPDATABLE);
String sql = "SELECT * FROM profile";
ResultSet resultSet = statement.executeQuery(sql);
resultSet.moveToInsertRow();
resultSet.updateString("id","123");
resultSet.updateString("username","test1");
resultSet.updateString("birthdate","1970-01-01");
resultSet.updateString("email","test1@jsp.com");
resultSet.insertRow();
```

JDBC ResultSet Update

- Updates existing data in the database
- Data is filtered / escaped by JDBC

```
Connection connection =
  DriverManager.getConnection(url,username,password)
Statement statement =
  connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                             ResultSet.CONCUR_UPDATABLE);
String sql = "SELECT * FROM profile WHERE id='123'";
ResultSet resultSet = statement.executeQuery(sql);
resultSet.next();
resultSet.updateString("username","test1");
resultSet.updateString("birthdate","1970-01-01");
resultSet.updateString("email","test1@jsp.com");
resultSet.updateRow();
```

JDBC Transactions

- connection.setAutoCommit(false) *(default: true)*
- connection.commit()
- connection.rollback()

JDBC Transaction Example

```
Connection connection = null;
try
{
  connection = getConnection();
  connection.setAutoCommit(false);
  runSomeQuery();
  runAnotherQuery();
  connection.commit();
}
catch (SQLException e)
{
  connection.rollback();
}
finally
{
  try
  {
    if (connection != null)
      connection.close();
  }
  catch (SQLException e)
  {
  }
}
```

JDBC MetaData

- Provides information about databases and drivers
- Commonly used to list tables, columns etc
- Comparable to introspection / Java reflection

Next Time

- Web Security