Managing Data on the World-Wide-Web

JDBC

Elad Kravi
Overview

1. RDBMS
2. JDBC
4. Advanced Aspects
Relational Database

• Was originally defined by Edgar Codd (IBM Research Center) in 1970

• A relation (represented as table) is defined as a set of tuples (rows) share the same attributes (columns).

• Relations can be joined to combine their data.
  – the combination is available by using shared attributes, also called primary keys
SQL (Structured Query Language)

- A query language designed for managing data in relational database management systems
  - initially developed at IBM by Donald D. Chamberlin and Raymond F. Boyce in the early 1970s
  - declarative language (rather than procedural)
    - describing what the program should accomplish, rather than how to go about accomplishing it
- Basic operations:
  - **Select** – retrieve data from the table
  - **Insert** – insert new row to the table
  - **Update** – update data in the table
  - **Delete** – delete row(s) from table

```sql
SELECT * FROM Book WHERE price > 100.00 ORDER BY title;
```
• Provides facilities for:
  – controlling data access
  – enforcing data integrity
  – managing concurrency control
  – recovering the database after failures and restoring it from backup files
  – maintaining database security
MySQL

- One of the world's most used relational database management system (RDBMS)
  - named after the developer’s daughter, My

- MySQL is written in C and C++. Its SQL parser is written in yacc and sql_lex.cc

- Has both textual and graphical interface
Overview

1. RDBMS
2. JDBC
4. Advanced Aspects
JDBC - Java Database Connectivity

• API that defines how a client may access a database
  – released by Sun Microsystems as part of JDK 1.1 on February 19, 1997.
  – currently being developed under the Java Community Process.
• Provides methods for querying and updating data in a database
• Information is transferred from relations to objects and vice-versa.
JDBC Architecture

These are Java classes

Network

Oracle Driver

DB2 Driver

Postgres Driver

Java Application

JDBC

Oracle

DB2

Postgres
Java code calls JDBC library

JDBC loads a driver
- A driver provides connection to a database
- It also converts JDBC calls to calls for a specific database

An application can work with several DBMS by using corresponding drivers

The API defined in the packages:
- Java.sql
- Javax.sql
Development Process

- Set up the driver
- Get Connection
- Submit SQL query
- Process result set
- Close resources
  - connections, statements, and result sets
In JDBC 4.0, the JDBC drivers are automatically loaded based on the classpath.

- JDBC 3.0 drivers have to be explicitly loaded:
  
  ```java
  Class.forName("com.mysql.jdbc.Driver");
  ```

When the driver is loaded, it's being registered to the DriverManager.

- The DriverManager mediates between the user and JDBC drivers
- It keeps track of the drivers that are available
- It also in charge of establishing a connection to the database (through the driver)
We provide an URL to tell the DriverManager which database to connect to

```java
Connection con = DriverManager.getConnection(url, username, password);
```

The URL is of the form

- `jdbc:<driver protocol>;<driver connection details>`

Examples:

- "jdbc:mysql://localhost:3306/mydb"
- "jdbc:oracle:thin:@myhost:1521:orcl"

The DriverManager looks for the driver that can talk to the corresponding database

- Set up the driver
- Get Connection
- Submit SQL query
- Process result set
- Close resources
import java.sql.*;
:

String url = "jdbc:mysql://localhost:3306/mydb";
String user = "user";
String password = "1234";

Connection conn =
    DriverManager.getConnection(url, user, password);
:

• An SQLException will be thrown in case database access error occurs
  – malformed URL
  – wrong credential
  – No suitable driver found

java.sql.SQLException: No suitable driver found for
jdbc:mysql://localhost:3306/mydb
Submitting an SQL Query

String url = "jdbc:mysql://localhost:3306/mydb";
String user = "user";
String password = "1234";

Connection conn =
    DriverManager.getConnection(url, user, password);

// We use the Connection object to create a statement:
Statement stmt = conn.createStatement();

// The actual execution:
ResultSet rs = stmt.executeQuery("select * from students");
• ResultSet objects provide access to the tables generated as results of executing Statement queries

• The table rows are retrieved in sequence:
  – a ResultSet object maintains a cursor pointing to its current row
  – Initially the cursor is positioned before the first row
  – next() advances the cursor to the next row and returns true if there are more rows to process

```
ResultSet rs = stmt.executeQuery("select * from students");

while (rs.next()) {
    String avg = rs.getString("average");
    :
}
```
• Only one ResultSet per Statement can be opened at a given time
  – All execution methods in the Statement interface implicitly close a statement's current ResultSet object if an open one exists
  – the result is saved in the memory and freed when rs.close() or stmt.close() are called
boolean res = stmt.execute(query);
int rowsModified = stmt.executeUpdate(query);

- Used for SQL statements such as `insert, delete, update` and `create table`
  - ‘execute’ returns true if the first result is a ResultSet object; false if it is an update count or there are no results
  - ‘executeUpdate’ returns the number of rows modified

```java
CREATE TABLE <table name>
...

INSERT INTO <table name>
VALUES (<values>)
```
try {

: 

} finally {
    if (stmt != null) {
        stmt.close();
    }

    if (conn != null) {
        conn.close();
    }
}
Timeout

```java
tstmt.setQueryTimeout(seconds);
```

- Sets a timeout for the driver to wait for a query to be completed.
  - if the operation is not completed in the given time, an `SQLException` is thrown

- What is it good for?
SQL injection & Prepared Statements

Statement `stmt = conn.createStatement();`

`ResultSet rs = stmt.executeQuery("SELECT PersonalData FROM MyTable WHERE name = '" + username + "'");`

// executeQuery() returns a single ResultSet object

- **What happens if:**

  `username = "' OR TRUE";`

• Solution:

```java
PreparedStatement pstmt =
    conn.prepareStatement("SELECT PersonalData FROM MyTable where name = ?");

pstmt.setString(1, username);
ResultSet rs = pstmt.executeQuery();
```

• The PreparedStatement object contains not just an SQL statement, but an SQL statement that has been precompiled.
  – treating the user information as data and not as part of the SQL
String updateString =
    "update products set price = ? where pid = ?";

PreparedStatement updatePrices =
    conn.prepareStatement(updateString);

for (Map.Entry<Integer, Integer> e : newPrices.entrySet()) {
    updatePrices.setInt(1, e.getValue().intValue());
    updatePrices.setInt(2, e.getKey());
    updatePrices.executeUpdate();
}
Overview

1. RDBMS
2. JDBC
4. Advanced Aspects
• How many connections should be kept?
  – (from MySql site) the number of connections permitted is controlled by the max_connections system variable.
  – how can we restrict this number on our own?

• Can we improve search by not going over all the rows?
  – we can use indices
Database Index

• A data structure (usually B-trees) that improves the speed of data retrieval operations on a database table.
• This is at the cost of slower writes and increased storage space.
• Can be defined over a column.
• Used by the DB on search queries.

```
CREATE [UNIQUE|FULLTEXT|SPATIAL]
INDEX index_name [index_type] ON
tbl_name (index_col_name,...)
[index_type]
```
Concurrency

• How do we handle multiple connections that may simultaneously modify the DB?
  – use transactions:
    A transaction comprises a unit of work performed within a DBMS against a database, and treated in a coherent and reliable way independent of other transactions
  – explained in the lectures
Hibernate is an Object/Relational Mapping (ORM) solution for Java environments.

Object/Relational Mapping:
- the technique of mapping data between an object model representation to a relational data model representation

Hibernate maps Java classes to database tables, and from Java data types to SQL data types.

http://hibernate.org/
Saving records:

```java
@Entity
@Table(name = "EVENTS")
public class Event {
    ...
}

Session session = sessionFactory.openSession();
session.beginTransaction();

session.save(new Event("Our very first event!", new Date()));
session.save(new Event("A follow up event", new Date()));
session.getTransaction().commit();

session.close();
```