Lecture 13: Database programming

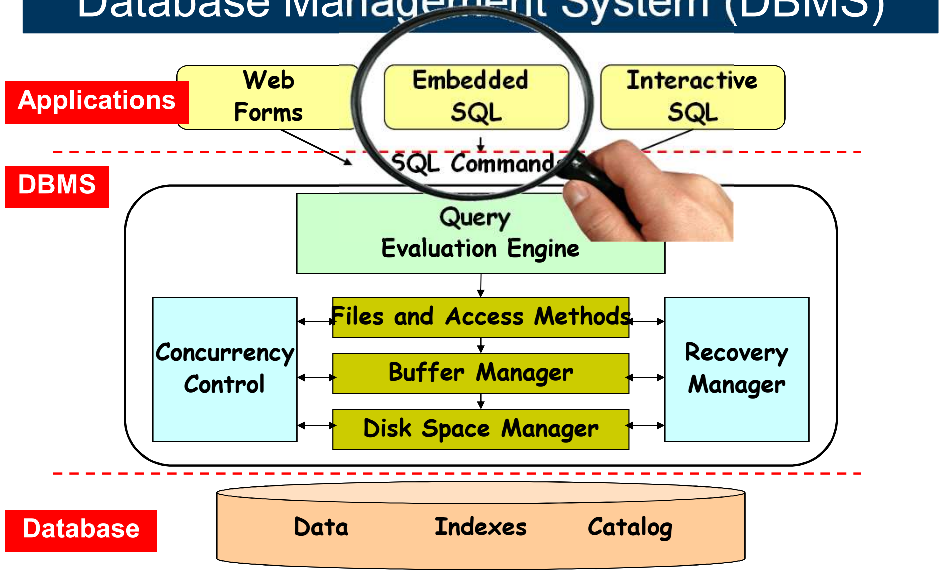
CS 1555: Database Management Systems

Constantinos Costa
http://db.cs.pitt.edu/courses/cs1555/current.term/

March 19, 2019, 16:00-17:15
University of Pittsburgh, Pittsburgh, PA

Lectures based: P. Chrysanthis & N. Farnan Lectures

Database Management System (DBMS)



Database Programming

- Objective:
  - To access a database from an **application** program (as opposed to interactive interfaces)
- Why?
  - An interactive interface is convenient but not sufficient
    - A majority of database operations are made thru application programs (increasingly thru web applications)

Database applications

- How can we realize applications that can wield DBMSs to address their data management needs?
  - Use a procedural language within the DBMS
    - PL/pgSQL
      - **P**rocedural **L**anguage extensions to **P**ost**g**re**SQL**
  - Write applications in a general purpose language
    - Embedded SQL
      - A statement-level database interface where SQL is written alongside *host language* code
    - Database access API
      - JDBC, ODBC, PHP

PostgreSQL Stored Procedures

```
CREATE [OR REPLACE] PROCEDURE name(parameters)
LANGUAGE language_name
AS $$
   stored_procedure_body;
$$;
```

- Unlike a user-defined function, a stored procedure does not have a **return** value.
  - If you want to end a procedure earlier, you can use the **RETURN** statement with no expression as follows: RETURN;

Stored Procedure Example

```
CREATE OR REPLACE PROCEDURE transfer(INT, INT, DEC)
LANGUAGE plpgsql
AS $$
BEGIN
   -- subtracting the amount from the sender's account
   UPDATE accounts
   SET balance = balance - $3
   WHERE id = $1;
   -- adding the amount to the receiver's account
   UPDATE accounts
   SET balance = balance + $3
   WHERE id = $2;
   COMMIT;
END;
$$
```

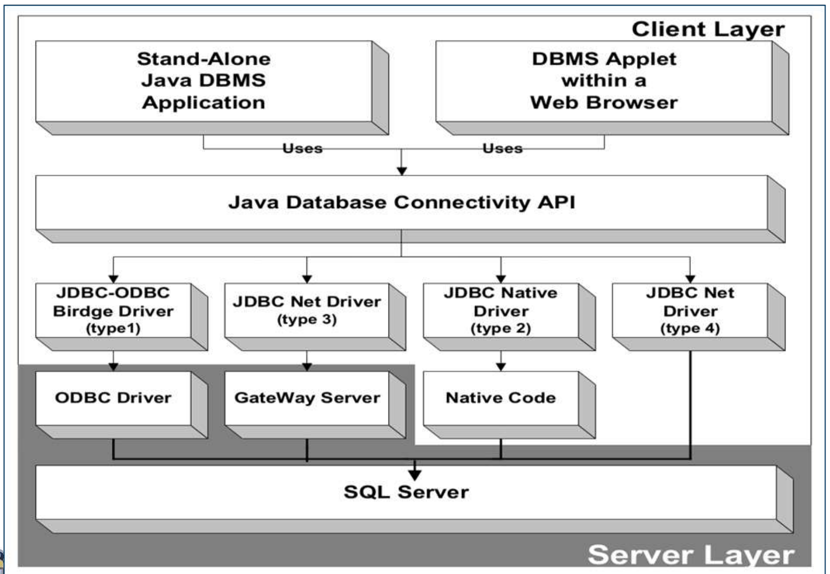
Advantages of stored procedures

- Reduce the number of round trips between applications and database servers.
- Increase application performance because the user-defined functions and stored procedure are pre-compiled and stored in the PostgreSQL database server.
- Reusable in many applications. Once you develop a function, you can reuse it in any applications.

Disadvantages of stored procedures

- Slowness in software development because stored procedure programming requires specialized skills that many developers do not possess.
- Difficult to manage versions and hard to debug.
- May not be portable to other database management systems e.g., MySQL or Microsoft SQL Server.

Database access API example: JDBC


Client Layer
Stand-Alone Java DBMS Application
DBMS Applet within a Web Browser
Uses
Uses
Java Database Connectivity API
JDBC-ODBC Birdge Driver (type1)
JDBC Net Driver (type 3)
JDBC Native Driver (type 2)
JDBC Net Driver (type 4)
ODBC Driver
GateWay Server
Native Code
SQL Server
Server Layer

JDBC Drivers

- Type 1
  - JDBC-ODBC bridge, translates JDBC calls into ODBC calls
- Type 2
  - Java JDBC Native Code, partial Java driver converts JDBC calls into client API for the DBMS
- Type 3
  - JDBC-Gateway, pure Java driver connects to a database middleware server that in turn interconnects multiple databases and performs any necessary translations
- Type 4
  - Pure Java JDBC. This driver connects directly to the DBMS

Useful Links

- JDBC DRIVER
  – https://jdbc.postgresql.org/download.html
- JDBC API
  – https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/

JDBC Example

```
package edu.pitt.cs;

import java.util.Properties;
import java.sql.*;

public class JavaDemo {
    public static void main(String args[]) throws
            SQLException, ClassNotFoundException {
        Class.forName("org.postgresql.Driver");
        String url = "jdbc:postgresql://localhost/postgres";
        Properties props = new Properties();
        props.setProperty("user", "postgres");
        props.setProperty("password", "password");
        Connection conn =
                DriverManager.getConnection(url, props);
```

JDBC Example

```
 Statement st = conn.createStatement();
        String query1 =
                "SELECT SID, Name, Major FROM CS1555.STUDENT
WHERE Major='CS'";
        ResultSet res1 = st.executeQuery(query1);
        int rid;
        String rname, rmajor;
        while (res1.next()) {
            rid = res1.getInt("SID");
            rname = res1.getString("Name");
            rmajor = res1.getString(3);
            System.out.println(rid + " " + rname + " " +
rmajor);
        }
    }
}
```

Moving cursors

```
package edu.pitt.cs;

import java.sql.*;
import java.util.Properties;

public class JavaDemoCursor {
    public static void main(String args[]) throws
            SQLException, ClassNotFoundException {
        Class.forName("org.postgresql.Driver");
        String url = "jdbc:postgresql://localhost/postgres";
        Properties props = new Properties();
        props.setProperty("user", "postgres");
        props.setProperty("password", "password");
        Connection dbcon =
                DriverManager.getConnection(url, props);

        Statement st = dbcon.createStatement(
                ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_READ_ONLY);
        ResultSet resultSet = st.executeQuery("SELECT * FROM CS1555.STUDENT");
```

Moving cursors

```
        int pos = resultSet.getRow();       // Get cursor position, pos = 0
        boolean b = resultSet.isBeforeFirst();    // true
        int rid;
        String rname, rmajor;

        resultSet.next();                   // Move cursor to the first row
        pos = resultSet.getRow();           // Get cursor position, pos = 1

        b = resultSet.isFirst();    // true
        resultSet.last();           // Move cursor to the last row
        pos = resultSet.getRow();   // If table has 10 rows, pos = 10

        b = resultSet.isLast();     // true
        resultSet.afterLast();      // Move cursor past last row
        pos = resultSet.getRow();   // If table has 10 rows,
        // value would be 11
        b = resultSet.isAfterLast();   // true

    }
}
```

Cursor Navigation Types

- Statement stC = dbcon.createStatement ({ResultSet.TYPE_XXXX});
- TYPE_XXXX
  - TYPE_FORWARD_ONLY: ResultSet can only be navigated forward.
  - SCROLL_INSENSITIVE: ResultSet can be navigated forward, backwards and jump. Concurrent db changes are not visible.
  - SCROLL_SENSITIVE: ResultSet can be navigated forward, backwards and jump. Concurrent db changes are visible.

Cursor Concurrency Types

- Statement stC = dbcon.createStatement (ResultSet.TYPE_XXXX);
- TYPE_XXXX
  - CONCUR_READ_ONLY: ResultSet can only be read
  - CONCUR_UPDATABLE: ResultSet can be updated

Bringing database values into host code

- get*X*()
  - E.g., getString(), getDouble(), getBoolean(), getBlob(), etc.
- getString(int columnIndex)
  - returns value from column# *columnIndex* of the current row
- getString(String columnLabel)
  - returns value from column with label *columnLabel* from the current row

How do we recognize NULL values?

- wasNull()
  - Returns a boolean
    - True if previous get*X*() call represents a NULL SQL value

Error handling

- Want to stick to Java's way of doing things

```
 try {
            ResultSet res3 = st.executeQuery("SELECT * FROM
NOTATABLE");
        }
        catch (SQLException e1) {
            System.out.println("SQL Error");
            while (e1 != null) {
                System.out.println("Message = " + e1.getMessage());
                System.out.println("SQLState = "+ e1.getSQLState());
                System.out.println("SQLState = "+ e1.getErrorCode());
                e1 = e1.getNextException();
            }
        }
```

Transactions

- Just like when interacting with PostgreSQL via psql, autocommit is enabled in JDBC
  - Each statement executed is treated as a transaction
  - Can be disabled with:
    - Connection.setAutoCommit(boolean autoCommit)
      - Note that changing the autocommit setting commits the current transaction
- With autocommit off, transactions can be completed with
  - Connection.commit()
  - Connection.rollback()

Transaction example

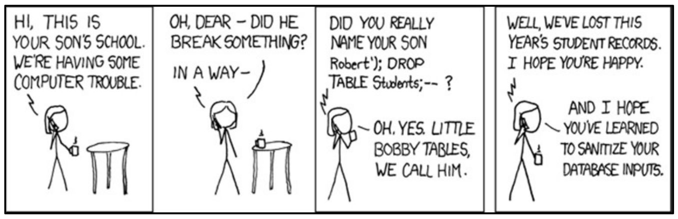
```
try {
    conn.setAutoCommit(false);
    st.executeUpdate("INSERT INTO ENROLLMENT VALUES (1,
'CS1501')");
    st.executeUpdate("INSERT INTO ENROLLMENT VALUES (1,
'CS1555')");
    conn.commit();
}
catch (SQLException e1) {
    try {
        conn.rollback();
    }
    catch(SQLException e2) {
System.out.println(e2.toString()); }
}
```

Consider the following:

```
Connection conn =
        DriverManager.getConnection(url, props);

String username="admin";
String password="1' OR '1'='1" ;
String sql = "SELECT * FROM cs1555.users WHERE username= '" +
username + "' and password='" + password + "'";
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
if (rs.next()) {
    loggedIn = true;
    System.out.println("Successfully logged in");
}
else {
    System.out.println("Username and/or password not
recognized");
}
```

SQL Injection


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Prepared statements

- Create and pre-compile parameterized queries using parameters markers, indicated by question marks (?)
  - E.g.,

    -
      ```
      PreparedStatement st2 = conn.prepareStatement(
          "SELECT * FROM STUDENTS WHERE Name = ?");
      ```
- Specify the values of parameters using set*X*(i,v)
  - i: argument-index
  - v: value
  - E.g.,

    -
      ```
      String fname = readString("Enter Name: ");
      st2.setString(1, fname);
      ResultSet rs2 = st2.executeQuery();
      ```
  - Can set NULL values with
    - PreparedStatement.setNull(int parameterIndex, int sqlType)

Fix SQL Injection

```
Connection conn =
        DriverManager.getConnection(url, props);

String username = "admin";
String password = "1' OR '1'='1";
PreparedStatement stmt = conn.prepareStatement("SELECT * FROM
cs1555.users WHERE username=? AND password=?");
stmt.setString(1, username);
stmt.setString(2, password);
ResultSet rs = stmt.executeQuery();
if (rs.next()) {
    loggedIn = true;
    System.out.println("Successfully logged in");
} else {
    System.out.println("Username and/or password not
recognized");
}
```