Outline

- Quiz 1
- Stored procedures
- Batch processing
- Transactions
- Homework #2
Quiz 1

- Ave: 9.25
- Mode: 10
- Min: 3
- Max: 15
Stored Procedures
Distributed Application

[Diagram showing a client program communicating with a database via JDBC and Driver]
Where to put computation?

- All client
  - use database just to store
  - copy to client to compute

- All server
  - use client just to display
  - use database for computation
<table>
<thead>
<tr>
<th></th>
<th>Data small</th>
<th>Data large</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Server available</strong></td>
<td>Either OK</td>
<td>Server</td>
</tr>
<tr>
<td><strong>Server busy</strong></td>
<td>Client</td>
<td>It depends</td>
</tr>
</tbody>
</table>
Stored Procedures

- Database-resident code
- Serve many purposes
  - computations too complex for SQL
  - triggers
  - move computation to the data
Steps

1. Write stored procedure
2. Create a callable statement
3. Set the parameters
4. Call the procedure
5. Possibly get return values
public static void updateAvailableRoomData
   (int hotelId, String roomType, int mumOfRooms, String updateFlag)
{
   Connection connection = null; // Database Connection Object
   try {
      connection = new OracleDriver().defaultConnection();
      String addBookedSeats = " + ";
      String subTotalSeats = " - ";
      if (updateFlag.equals("COUT"))
      {
         addBookedSeats = " - ";
         subTotalSeats = " + ";
      }
      if (roomType.equals("ORCL")) roomType = "OTHR";
      PreparedStatement pstmt =
         connection.prepareStatement("UPDATE ROOM_AVAILABILITY " + " SET BOOKED_" +
         roomType + " = BOOKED_" + roomType + " + addBookedSeats + mumOfRooms + " , TOTAL_"
         + roomType + " = TOTAL_" + roomType + " + subTotalSeats + mumOfRooms + " WHERE
         HOT_ID = ? AND BOOKING_DATE = " + " ( SELECT MAX(BOOKING_DATE) FROM
         ROOM_AVAILABILITY " + " WHERE HOT_ID = ? )" );
      pstmt.setInt(1,hotelId); // Bind the Hotel ID input parameter
      pstmt.setInt(2,hotelId); // Bind the Hotel Id input parameter int
      noRecordsUpdated = pstmt.executeUpdate(); // Execute the Statement
      ... et cetera ...
From client

```java
CallableStatement stmt = con.prepareStatement("Call Hotel_HotelBookingsSample_updateAvailableRoomData (?, ?, ?, ?)");

stmt.setInt (1, 5);
stmt.setString (2, "Single");
stmt.setInt (3, 1);
stmt.setString (4, "COUT");

stmt.execute();
```
Return values?

```java
stmt.registerOutParameter (3, Types.INTEGER);
stmt.execute();
int result = Stmt.getInt(3);
```
Summary – Stored Procedures

- Embed computation in database using database-specific code

  - benefits
    - move computation to data

  - drawbacks
    - SP code not portable
    - maintenance problematic

- still frequently used for performance benefits
JDBC Classes

- DriverManager
- Connection
- Statement
- PreparedStatement
- CallableStatement
- ResultSet
- Data Types

- DriverManager : getConnection -> Connection
- DriverManager : prepareCall -> CallableStatement
- DriverManager : preparedStatement -> PreparedStatement
- DriverManager : createStatement -> Statement
- DriverManager : createStatement -> PreparedStatement
- DriverManager : createStatement -> ResultSet

- Connection : getConnection -> Connection
- Connection : preparedStatement -> PreparedStatement
- Connection : createStatement -> Statement
- Connection : createStatement -> PreparedStatement
- Connection : createStatement -> ResultSet

- PreparedStatement : setXXX, executeQuery -> ResultSet
- PreparedStatement : setXXX, executeQuery -> ResultSet
- PreparedStatement : executeQuery -> ResultSet
- PreparedStatement : executeQuery -> ResultSet

- CallableStatement : setXXX, getXXX
- CallableStatement : setXXX, getXXX
- CallableStatement : setXXX, getXXX
- CallableStatement : setXXX, getXXX

- ResultSet : getXXX
- ResultSet : getXXX
Batch Updating
Limit Data Movement

- Cost of moving data between client and server
- One solution
  - Stored procedure: move computation to data
- Another
  - Try to reduce the cost by reducing overhead
  - Communication costs are “chunky”
Non-linear Communication Costs
Example

- Suppose 12 small operations
- Execution time
  - $= 12 \times (\text{op\_time} + \text{comm\_in} + \text{comm\_out})$
- If grouped together
  - $= 12 \times \text{op\_time} + \text{comm\_in} + \text{comm\_out}$
Steps

1. Turn off auto commit
2. For each update, call addBatch instead of executeUpdate
3. when complete, call executeBatch
con.setAutoCommit(false);
...
JDBC calls ...
Statement stmt = con.prepareStatement ("INSERT ...;");
...
set parameters ...
// ready to update
stmt.addBatch();  // instead of executeUpdate()
...
more stuff ...
stmt2.addBatch ();
...
more stuff ...
stmt3.addBatch ();
// updates complete
con.executeBatch();
con.executeBatch();
con.setAutoCommit(true);
Summary - Batch Processing

- Useful when there are many updates
- Benefits
  - Lower communication overhead to database
- Problems
  - Each statement treated separately
    - No chance to handle exceptions that affect later updates
Transactions

- Goal
  - In the face of machine failure, keep the database consistent
Transactions

- Atomicity
  - No partial success
- Consistency
  - End point is consistent
- Isolation
  - Transaction invisible until completed
- Durability
  - Committed actions survive system failure
Example

- Reservation is added

- Actions
  - Record inserted into Reservation table
  - Capacity table updated with new reservation count
Steps

1. Set auto commit to false
2. perform updates as normal
3. Commit transaction
4. Set auto commit back to true
If there is a failure, rollback the transaction
   - typically exception handler
Example

```java
try
{   conn.setAutoCommit (false);
    addReservation (conn);
    updateCapacity (conn);
    conn.commit ();
} catch (SQLException e)
{   try
    {   conn.rollback ();
        DBUtilities.displaySQLException (e);
    } catch (SQLException e2) { DBUtilities.displaySQLException (e2); } 
} finally
{   try
    {   conn.setAutoCommit (true);
        DBUtilities.displaySQLException (e3);
    } catch (SQLException e3) { DBUtilities.displaySQLException (e3); } 
}
```
Homework #2
Homework #2

- Frequent Flyer Program
- Database
- Interactive text mode
% java -cp "." FFTextMode
Setting up connection to jdbc:odbc:hwk2
FF: level
Enter benefit level: 1
Id   Name            Since   Level
 5   Rubble, Barney   02/01/01 1
 9   Mouse, Minnie   02/03/85 1
10   Spratt, Jack    11/11/99 1
FF: add-flight
Enter flight id: 2032
Enter flight no: 359
Enter date (mm/dd/yy): 3/4/03
Enter origin: ORD
Enter destination: PHX
Enter miles: 1500
Enter member ids: (-1 to halt) 2
Enter member ids: (-1 to halt) 8
Enter member ids: (-1 to halt) -1
FF: exit
Commands

- add-flight
- find-by-level
- help
- exit
- flights
- delete-flight
- find by name
- members
How to design?
Adaptability

- What are the most likely vectors of change?
Command Pattern

- Represent users action with an object
- Framework in which command objects are created and executed
Central Loop

- Loop
  - Create command object
  - Set parameters
  - Execute
Command Line Interaction

% java FFTxtMode delete-flight 37
%

Command Hierarchy
Advice

- Start right away
- Don’t use JDK 1.3
- Close statements and result sets
- Start right away
Programming example

- Exit command
- ShowFlights command