

COMP67321

Transactions in Relational Databases

Goran Nenadic

School of Computer Science
University of Manchester

1

Aims

- Learn and understand basic concepts of transactional processing,
- Review Java Database Connectivity (JDBC)

2

Transactions

- A transaction is a sequence of queries and update statements executed as a single unit
- Transactions are started implicitly and terminated by one of the following:
 - commit work**: makes all updates of the transaction permanent in the database
 - rollback work**: undoes all updates performed by the transaction.

3

continued

Transactions

- Example
 - transfer of money from one account to another involves two steps: deduct from one account and credit to another
 - if one steps succeeds and the other fails, database is in an inconsistent state
 - therefore, either both steps should succeed or neither should
- If any step of a transaction fails, all work done by the transaction can be undone by **rollback**
- Rollback of incomplete transactions is done automatically, in case of system failures

4

continued

Transactions

- In most database systems, each SQL statement that executes successfully is automatically committed.
 - each transaction would then consist of only a single statement
 - automatic commit can usually be turned off, allowing multi-statement transactions, but how to do so depends on the database system

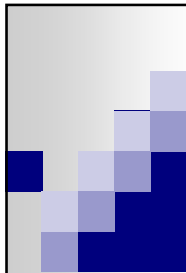
5

continued

Transactions

- Another option in SQL:1999: enclose statements within
 - begin atomic**
 - ...
 - end**

6



JDBC: Programming Relational Databases from Java

7

JDBC and SQLJ

- There are two standard interfaces allowing relational databases to be accessed and manipulated from Java:
 - *JDBC*: a class library that allows dynamic SQL statements to be called from Java.
 - *SQLJ*: a pre-processor that allows static SQL statements to be embedded in Java.
- JDBC is much more widely used.

8

JDBC

- JDBC is a Java API for communicating with database systems supporting SQL
- JDBC supports a variety of features for querying and updating data, and for retrieving query results
- JDBC also supports metadata retrieval, such as querying about relations present in the database and the names and types of relation attributes

9

continued

JDBC

- JDBC can be used in client applets or applications, or (in some database systems) for implementing server-side functionality.
- JDBC involves no extensions to the syntax of Java. The JDBC package is imported thus:

```
import java.sql.*
```
- Specific database systems are accessed using vendor or third party drivers:

```
DriverManager.registerDriver(  
    new oracle.jdbc.driver.OracleDriver());
```

10

continued

JDBC

- Model for communicating with the database:
 - Open a connection
 - Create a "statement" object
 - Execute queries using the Statement object to send queries and fetch results
 - Exception mechanism to handle errors

11

Connecting to a database

- Statements and transactions are associated with connections.
- There are several ways of establishing a connection.
- Example

```
String url =  
    "jdbc:oracle:thin:@sr.cs.man.ac.uk:1526:teach";  
Connection conn = DriverManager.getConnection  
    (url,username,password);
```

12

Single-slide example

```
import java.sql.*;
class Trains
{ public static void main (String args [])
  throws SQLException
  {
    DriverManager.registerDriver(...);
    String url = "...";
    Connection conn = DriverManager.getConnection
      (url,args[0],args[1]);
    Statement stmt = conn.createStatement();
    ResultSet rset = stmt.executeQuery
      ("select T# from TRAIN");
    while (rset.next())
      System.out.println (rset.getString(1));
  }
}
```

13

Query results

- The result of executing a query is a `ResultSet`, which supports:
 - iterator functionality, through `boolean next()`, `boolean previous()`.
 - update functionality, for results from simple queries, through `deleteRow()`, `updateXXX()`.
 - control functionality, through `setFetchSize(int rows)`
 - tuple access functionality (see next)

14

Accessing result tuples

- In JDBC there is no predefined Java type for the result of a query, so attribute values are retrieved by:
 - `getXXX()` functions, where `XXX` is the result type.
- The argument to the function is either the column position (starting from 1) or its name.
- Note the potential for runtime errors if the result is not as anticipated.

15

example

JDBC – code example

```

public static void JDBCexample(String dbid, String userid,
                               String passwd)
{
    try {
        Class.forName ("oracle.jdbc.driver.OracleDriver");
        Connection conn = DriverManager.getConnection(
            "jdbc:oracle:thin:@aura.bell-labs.com:2000:bankdb", userid,
            passwd);
        Statement stmt = conn.createStatement();
        ... Do Actual Work ....
        stmt.close();
        conn.close();
    }
    catch (SQLException sqle) {
        System.out.println("SQLException : " + sqle); }
}

```

16

Prepared statements

- A *PreparedStatement* object allows an SQL statement to be run multiple times, with different parameters, without the SQL being recompiled by the database.
- Simple example:

```

PreparedStatement pstmt = conn.prepareStatement(
    "select t# from train where source = ?");
pstmt.clearParameters();
pstmt.setString(1,args[2]);
ResultSet rset = pstmt.executeQuery();

```

17

continued

Prepared statements

- Creating a prepared statement – formal parameters are identified by “?”s:


```

PreparedStatement pstmt =
conn.prepareStatement(
    "insert into booking values (?,?,?)");

```
- Parameters are bound using `setXXX (pos, val)` (`pos` starts from 1):


```

pstmt.setString(1,args[2])

```
- The request is executed using `executeQuery()` or `executeUpdate()`.

18

example

JDBC – code example

- Update to database

```
try {
    stmt.executeUpdate( "insert into account values
                        ('A-9732', 'Perryridge', 1200)");
} catch (SQLException sqle) {
    System.out.println("Could not insert tuple. " + sqle);
}
```

19

example

JDBC – code example

- Execute query and fetch and print results

```
ResultSet rset =
    stmt.executeQuery( "select branch_name, avg(balance)
                      from account
                      group by branch_name");
while (rset.next()) {
    System.out.println(
        rset.getString("branch_name") + " " +
        rset.getFloat(2));
}
```

20

example

JDBC – code example

- Some details
 - Getting result fields: rs.getString("branchname") and rs.getString(1) equivalent if branchname is the first argument of the select result.
 - Dealing with Null values

```
int a = rs.getInt("a");
if (rs.isNull()) Systems.out.println("Got null value");
```

21

Transactions in JDBC

- By default, each statement executes in a distinct transaction.
- To group statements, where `conn` is a `Connection`, use:
 - `conn.setAutoCommit(false)` to override the single-statement default and start a transaction.
 - `conn.commit()` and `conn.rollback()` to complete a transaction.

22

Closing things down

- The `close()` operation is supported on lots of things:
 - `Connection`
 - `Statement`
 - `ResultSet`
- In all cases, `close()` reclaims resources; it is good practice to close all the above as soon as possible.

23

Handling Errors – Important!

```
Connection conn = null;
try { ...
} catch (SQLException e) {
    System.out.println("SQL Exception: " +
        e.getMessage());
} finally {
    if (conn != null) {
        try {
            conn.rollback(); conn.close();
        } catch (SQLException sqlEx) { // ignore
        }
    }
}
```

24

Summary

- Transactions are used to control a set of queries, in particular updates
- SQL commands can be invoked from host languages.
 - JDBC is an example of a database API to access SQL databases from Java (most-widely used)
 - JDBS supports dynamic SQL (runtime errors?)
 - Similar APIs for most languages

25

Further Reading

- Oracle 10g JDBC Developers Guide and Reference, 2001 [Chapter 1: Overview; Chapter 3: Basic Features].
- Sun JDBC Tutorial:
 - <http://java.sun.com/docs/books/tutorial/jdbc/>

26
