JDBC:
Java Database Connectivity

1

Introduction to JDBC

- JDBC is used for accessing databases from Java applications
- Information is transferred from relations to objects and vice-versa
  - databases optimized for searching/indexing
  - objects optimized for engineering/flexibility

2

Why Access a Database with Java?

- There are queries that can not be computed in SQL:
  - Given a table **Bus(Source, Destination)** find all pairs of places that it is possible to travel (paths of any length)
- Java allows for a convenient user interface to the database

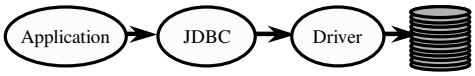
3

Six Steps

- Load the driver
- Establish the Connection
- Create a Statement object
- Execute a query
- Process the result
- Close the connection

4

JDBC Architecture



- Java code calls JDBC library
- JDBC loads a *driver*
- Driver talks to a particular database
- Can have more than one driver -> more than one database
- Ideal: can change database engines without changing any application code

5

Loading the Driver

- We can register the Driver indirectly using the Java statement:

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

  Calling `Class.forName`, automatically
  - creates an instance of the driver
  - registers the driver with the `DriverManager`
- The DriverManager tries all the drivers
- Uses the first one that works

6

Packages to Import

- In order to connect to the Oracle database from java, import the following packages:
  - java.sql.*; (usually enough)
  - javax.sql.* (for advanced features, such as scrollable result sets)

7

Connecting to the Database

```
String path = "jdbc:oracle:thin:";
String host = "sol4";
String port = "1521";
String db = "stud";
String login = "snoopy";
String url = path + login + "/" + login +
        "@" + host +":" + port + ":" + db;

Class.forName("oracle.jdbc.driver.OracleDriver");

Connection con =  DriverManager.getConnection(url);
```

This is actually the password

8

Connection Methods

Statement createStatement()
- returns a new Statement object

PreparedStatement prepareStatement(String sql)
- returns a new PreparedStatement object

CallableStatement prepareCall(String sql)
- returns a new CallableStatement object

- Why all these different kinds of statements? Optimization.

Querying with Statement

```
String queryStr =
      "SELECT * FROM Sailors " +
      "WHERE Lower(Name) = 'joe smith'";

Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery(queryStr);
```

- Statements are used for queries that are only issued once.
- The executeQuery method returns a ResultSet object representing the query result.

10

Changing DB with Statement

```
String deleteStr =
      "DELETE FROM Sailors " +
      "WHERE sid = 15";

Statement stmt = con.createStatement();
int delnum = stmt.executeUpdate(deleteStr);
```

- executeUpdate is used for data manipulation: insert, delete, update, create table, etc. (anything other than querying!)
- executeUpdate returns the number of rows modified.

11

About Prepared Statements

- Prepared Statements are used for queries that are executed many times.
- They are parsed only once.
- Using setString(i, value) (setInt(i, value), etc.) the i-th question mark is set to the given value.

12

Querying with PreparedStatement

```
String queryStr =
      "SELECT * FROM Sailors " +
      "WHERE Name = ? and Rating < ?";

PreparedStatement pstmt =
      con.prepareStatement(queryStr);

pstmt.setString(1, "Joe");
pstmt.setInt(2, 8);

ResultSet rs = pstmt.executeQuery();
```

13

Changing DB with PreparedStatement

```
String deleteStr =
      "DELETE FROM Boats " +
      "WHERE Name = ? and Color = ?";

PreparedStatement pstmt =
      con.prepareStatement(deleteStr);

pstmt.setString(1, "Fluffy");
pstmt.setString(2, "red");

int delnum = pstmt.executeUpdate();
```

14

Statements vs. PreparedStatements: Be Careful!

- Are these the same? What do they do?

```
String val = "Joe";
PreparedStatement pstmt =
      con.prepareStatement("select * from Sailors
where sname=?");
pstmt.setString(1, val);
ResultSet rs =  pstmt.executeQuery();
```

```
String val = "Joe";
Statement stmt =  con.createStatement( );
ResultSet rs =  stmt.executeQuery("select * from
Sailors where sname=" + val);
```

15

Statements vs. PreparedStatements: Be Careful!

- Will this always work?

```
Statement stmt =  con.createStatement( );
ResultSet rs =  stmt.executeQuery("select * from R
where A ='" + val + "' ");
```

- The moral: When getting input from the user, always use a PreparedStatement

16

Statements vs. PreparedStatements: Be Careful!

- Will this work?

```
PreparedStatement pstmt =
      con.prepareStatement("select * from ?");

pstmt.setString(1, "Sailors");
```

17

ResultSet

- A ResultSet provides access to a table of data generated by executing a Statement.
- Only one ResultSet per Statement can be open at once.
- The table rows are retrieved in sequence.
- A ResultSet maintains a cursor pointing to its current row of data.
- The 'next' method moves the cursor to the next row.

ResultSet Methods

- *Type* get *Type*(int columnIndex)
  - returns the given field as the given type
  - fields indexed starting at 1 (not 0)
- *Type* get *Type*(String columnName)
  - same, but uses name of field
  - less efficient
- int findColumn(String columnName)
  - looks up column index given column name

isNull

- In SQL, NULL means the field is empty
- Not the same as 0 or ""
- In JDBC, you must explicitly ask if a field is null by calling ResultSet.isNull(column)

Printing Query Output: Result Set (1)

Print Column Headers:

```
ResultSetMetaData rsmd = rs.getMetaData();
int numcols = rsmd.getColumnCount();

for (int i = 1 ; i <= numcols; i++) {
      if (i > 1) System.out.print(",");
      System.out.print(rsmd.getColumnLabel(i));
}
```

21

Printing Query Output: Result Set (2)

```
while (rs.next()) {
      for (int i = 1 ; i <= numcols; i++) {
            if (i > 1) System.out.print(",");
            System.out.print(rs.getString(i));
      }
      System.out.println("");
}
```

- To get the data in the i-th column: rs.getString(i)
- To get the data in column Abc: rs.getString("Abc")

22

Mapping Java Types to SQL Types

SQL type	Java Type
CHAR, VARCHAR, LONGVARCHAR	String
NUMERIC, DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT, DOUBLE	double
BINARY, VARBINARY, LONGVARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

Cleaning Up After Yourself

- Remember to close the Connections, Statements, PreparedStatements and ResultSets

```
con.close();
stmt.close();
pstmt.close();
rs.close()
```

24

Dealing With Exceptions

- A exception can have more exceptions in it.

```
catch (SQLException e) {
   while (e != null) {
        System.out.println(e.getSQLState());
        System.out.println(e.getMessage());
        System.out.println(e.getErrorCode());
        e = e.getNextException();
   }
}
```

25

Timeout

- Use setQueryTimeOut(int seconds) to set a timeout for the driver to wait for a statement to be completed
- If the operation is not completed in the given time, an SQLException is thrown
- What is it good for?

26

Advanced Topics

27

Transactions

- Transaction = more than one statement which must all succeed (or all fail) together
- If one fails, the system must reverse all previous actions
- Also can't leave DB in inconsistent state halfway through a transaction
- COMMIT = complete transaction
- ROLLBACK = abort

28

Example

- Suppose we want to transfer money from bank account 13 to account 72:

```
PreparedStatement pstmt =
      con.prepareStatement("update BankAccount
                            set amount = amount + ?
                            where accountId = ?");
pstmt.setInt(1,-100);
pstmt.setInt(2, 13);
pstmt.executeUpdate();
pstmt.setInt(1, 100);
pstmt.setInt(2, 72);
pstmt.executeUpdate();
```

What happens if this update fails?

29

Transaction Management

- The connection has a state called AutoCommit mode
- if AutoCommit is true, then every statement is automatically committed
- if AutoCommit is false, then every statement is added to an ongoing transaction
- Default: true

30

AutoCommit

Connection.setAutoCommit(boolean val)

- If you set AutoCommit to false, you must explicitly commit or rollback the transaction using Connection.commit() and Connection.rollback()

31

Fixed Example

```
con.setAutoCommit(false);
try {
  PreparedStatement pstmt =
      con.prepareStatement("update BankAccount
                            set amount = amount + ?
                            where accountId = ?");
  pstmt.setInt(1,-100); pstmt.setInt(2, 13);
  pstmt.executeUpdate();
  pstmt.setInt(1, 100); pstmt.setInt(2, 72);
  pstmt.executeUpdate();
  con.commit();
catch (Exception e) {
  con.rollback();
}
```

2