|  |  |
| --- | --- |
| sait_icon_wordmark_horiz_text_black | **CPRG 307**  **JDBC Student Handout** |

Java Database Connectivity (JDBC) is a standard API that enables Java programs to connect and access databases. The API specification does not change between RDBMS vendors, however, the driver does (and possibly the SQL syntax), so make sure you have the correct driver in your CLASSPATH.

When using this package import “java.sql.\*”.

The first step in any application that will access the database is creating the actual connection. This connection will “log” the user into the database so the user name and password are required – must be a user that exists in the RDBMS. In most cases, this connection should be made only once in an application rather than each time a database request is made.

For the purposes of this class, creating one class that handles the JDBC calls is ideal with a class level private variable of type Connection. The Connection variable will hold the database connection information which can then be utilized by all other methods within the class and requiring only one call to the actual method that connects to the database.

Connection:

* Initialize and load into memory the jdbc driver class (place in a try / catch block). This command will be specific for the RDBMS being used.
  + Example: Class.forName("oracle.jdbc.driver.OracleDriver");
* Connect to the database (place in a try / catch block).
  + Example: conn = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE",userName, password);
  + “conn” is the private Connection variable.
  + “userName” and “password” should not be hard coded, prompt the user for these values.
  + “ICT158470.ACDM.DS.SAIT.CA” ICT158470.ACDM.DS.SAIT.CA"ock)pt the user for these valuesactual method that connects to the database.e utilized by all other mis the name of the machine where the RDBMS is located. This will be machine specific.
  + “XE” is the name of the database installed with the express edition.
* Closing the connection (place in a try / catch block in a separate method).
  + Upon leaving the application or if there is a log out function, the database connection should be closed.
  + Example: conn.close();

DML:

* Commits / Rollbacks
  + By default, the JDBC connection commits all DML transactions automatically. This can be changed.
  + Example: conn.setAutoCommit(false);
  + Once the auto commit is turned off, commits and rollbacks can be done manually.
  + Example: conn.commit();
  + Example: conn.rollback();
* Performing a database update.
  + Create a statement.
  + Example: Statement stmt = conn.createStatement();
  + Execute the statement against the database.
  + Example: stmt.executeUpdate(updateCmd);
    - Where “updateCmd” is the SQL command.
  + Close the statement.
  + Example: stmt.close();
  + Put all of this in a try catch block. Your exception block should print the stack so the exact message can be captured. Generally, it will be an error directly from the database that will hold exact details of the problem encountered.
* Performing a database insert.
  + Create a statement.
  + Example: Statement stmt = conn.createStatement();
  + Execute the statement against the database.
  + Example: stmt.executeUpdate(insertCmd);
    - Where “insertCmd” is the SQL command.
  + Close the statement.
  + Example: stmt.close();
  + Put all of this in a try catch block. Your exception block should print the stack so the exact message can be captured. Generally, it will be an error directly from the database that will hold exact details of the problem encountered.
* Performing a database delete.
  + Create a statement.
  + Example: Statement stmt = conn.createStatement();
  + Execute the statement against the database.
  + Example: stmt.executeUpdate(deleteCmd);
    - Where “deleteCmd” is the SQL command.
  + Close the statement.
  + Example: stmt.close();
  + Put all of this in a try catch block. Your exception block should print the stack so the exact message can be captured. Generally, it will be an error directly from the database that will hold exact details of the problem encountered.

DQL (Queries):

* Queries are different than the DML commands in that data (a result set) is returned to the calling routine and needs to be parsed.
* DQL statement.
  + Create a statement.
  + Example: Statement stmt = conn.createStatement();
  + Execute the statement against the database.
  + Example: ResultSet rs = stmt.executeUpdate(queryStatement);
    - Where “queryStatement” is the SQL command.
  + Parse the statement.
  + Close the statement.
  + Example: stmt.close();
  + Put all of this in a try catch block. Your exception block should print the stack so the exact message can be captured. Generally, it will be an error directly from the database that will hold exact details of the problem encountered.
* Parsing the result set.
  + Parse each row at a time using a loop and a string tokenizer.
  + When creating the tokenizer, the first parameter will be a list of column headings that will be returned with the query separated by commas. If you used column alias’ in your query, remember these become your column headings.
  + Example: StringTokenizer st = new StringTokenizer(columnHeaders, ",");
    - Where “columnHeaders” is a string containing the column names separated by commas.
  + The parsed results can then be returned to the calling routine.