Working with Apache Derby

Connecting to the database

Loading the JDBC driver

Before you can use a database you must first load its JDBC driver.

```
// This is the Embedded JDBC Driver for the Derby database.
String driver = "org.apache.derby.jdbc.EmbeddedDriver";
// Load the driver to start up Derby.
Class.forName(driver).newInstance();
```

Connecting to the database

Once the JDBC driver has been loaded, you can use a DriverManager to connect to the database using a database URL.

```
// URL connection protocol to use, in this case Derby via JDBC
String protocol = "jdbc:derby:";
// Path for the database directory
String databaseDir = "derbyDB";
// Specify a username/password that can be used for connecting to the database.
String user = "smith";
String pass = "drowssap";
// Create a connection to the database. Specifying create=true causes a new
// database to be created if it does not yet exist. To remove the database,
// remove the database directory and its contents.
Connection connection = DriverManager.getConnection(protocol + databaseDir +
";create=true", user, pass);
```

The full connection string would look like this:

```
jdbc:derby:derbyDB;create=true
```

Shutting down the database

If the database is no longer needed, the database should be shut down so it can perform a checkpoint and release its resources. This step is not necessarily vital, but the database will take a bit longer to connect next time because it will run its recovery code.

```
DriverManager.getConnection("protocol + databaseDir + ";shutdown=true");
```

If you omit the database path, all Derby databases will shutdown. In other words:

```
jdbc:derby:derbyDB;shutdown=true  // shuts down the derbyDB database
jdbc:derby:;shutdown=true         // shuts down all databases
```

Using the database

You use the database via its Connection object. You may want to set the connection to commit changes manually rather than automatically.

```
Connection.setAutoCommit(false);

connection.commit();    // Commits a change
connection.rollback();  // Aborts a change
```

Statements

You issue commands to the database via statements obtained through the Connection object.

```
Statement statement = connection.createStatement();
statement.execute("...");  // SQL command
```

Creating tables

Before creating or even using tables within the database, you may want to see if they exist first. You can do this using the DatabaseMetaData interface.

```
DatabaseMetaData dbmd = connection.getMetaData();
ResultSet results = dbmd.getTables(null, null, "PERSON_TABLE", null);
if (results.next() && results.getString(3).equals("PERSON_TABLE"))
{ /* The table already exists. */ }
else
{ /* The table does not exist; create it. */ }
```

A table can be created as follows. Available data types can be found in the Derby Reference Manual.

```
statement.execute("CREATE TABLE PERSON_TABLE (" +
"PERSON_TABLE_SEQ_ID  BIGINT  NOT NULL " +
"  GENERATED ALWAYS AS IDENTITY " +
"  CONSTRAINT PK_PERSON_TABLE_SEQ_ID Primary Key, " +
"FIRST_NAME  VARCHAR(30)  NOT NULL, " +
"LAST_NAME   VARCHAR(30)  NOT NULL, " +
"AGE         SMALLINT     NOT NULL" + ")");
```

A column can be specified as a primary key adding CONSTRAINT constraint-name PRIMARY KEY after its definition. Similarly, a column can be made unique by specifying CONSTRAINT constraint-name UNIQUE. You define sequences by specifying GENERATED ALWAYS AS IDENTITY. You can also specify at what number the sequencing should start and even how much to it by specifying something like GENERATED ALWAYS AS IDENTITY START WITH 10 INCREMENT BY 2.

You can add foreign keys by altering an existing table.

```
statement.execute("ALTER TABLE OTHER_TABLE " +
"ADD CONSTRAINT FK_PERSON_ID " +
"Foreign Key(PERSON_ID) REFERENCES PERSON_TABLE(PERSON_TABLE_SEQ_ID)");
```

Executing statements

For often used SQL commands, you may want to use PreparedStatements.

```
PreparedStatement preparedStatement = connection.prepareStatement("...");
```

If you are inserting data into a table that has a sequenced column and if you would like to get the value of the resulting sequence number, you can specify a PreparedStatement as follows.

```
PreparedStatement preparedStatement = connection.prepareStatement("...",
Statement.RETURN_GENERATED_KEYS);
```

When inserting data into a table, you can use wildcards in your SQL statement.

```
PreparedStatement addNewPersonEntry = connection.prepareStatement(
"INSERT INTO PERSON_TABLE (FIRST_NAME, LAST_NAME, AGE) " +
"VALUES (?,?,?)", Statement.RETURN_GENERATED_KEYS);
```

You would prepare and execute this statement as follows.

```
addNewPersonEntry.setString(1, "Joe");     // FIRST_NAME
addNewPersonEntry.setString(2, "Smith");   // LAST_NAME
addNewPersonEntry.setShort(3, 25);         // AGE
addNewPersonEntry.execute();
```

To get the generated sequence value, you could do the following.

```
ResultSet results = addNewPersonEntry.getGeneratedKeys();
if (results.next())
{ Long personID = results.getLong(1); }  // PERSON_TABLE_SEQ_ID
```

References

Apache Derby website – http://db.apache.org/derby/
Derby Reference Manual – http://db.apache.org/derby/docs/dev/ref/

Tables
Create Table – http://db.apache.org/derby/docs/dev/ref/rrefsqlj24513.html
Data types – http://db.apache.org/derby/docs/dev/ref/crefsqlj31068.html#crefsqlj31068
Primary Key/Unique – http://db.apache.org/derby/docs/dev/ref/rrefsqlj16095.html#rrefsqlj16095
Foreign Keys – http://db.apache.org/derby/docs/dev/ref/rrefsqlj42154.html
Sequences – http://db.apache.org/derby/docs/dev/ref/rrefsqlj37836.html#rrefsqlj37836
Delete Table – http://db.apache.org/derby/docs/dev/ref/rrefsqlj34148.html

Common SQL Commands
Select – http://db.apache.org/derby/docs/dev/ref/rrefsqlj41360.html
Insert – http://db.apache.org/derby/docs/dev/ref/rrefsqlj40774.html
Update – http://db.apache.org/derby/docs/dev/ref/rrefsqlj26498.html
Delete – http://db.apache.org/derby/docs/dev/ref/rrefsqlj35981.html