# CO3409 Distributed Enterprise Systems

# Persistence Using a Database and JSFs

## Summary

This exercise looks at an interface between a JSF frontend and a database. However, the most important parts of the exercise are the use of the IDE to create and populate a database, to generate a class corresponding to a record from the database, and the use of the Java Persistence API to read and write records from Java.

## Purpose

On completion of this you should be able to

1. Use NetBeans to create a database web application
2. Create a simple class from a database using an object-relational mapping
3. Create and view a database using the NetBeans IDE
4. Load and save entities programmatically

## Activities

This tutorial consists of 4 parts.

### 1. Creating a Database using the NetBeans IDE

Creating a Database using Netbeans is straightforward. Follow the instructions below.

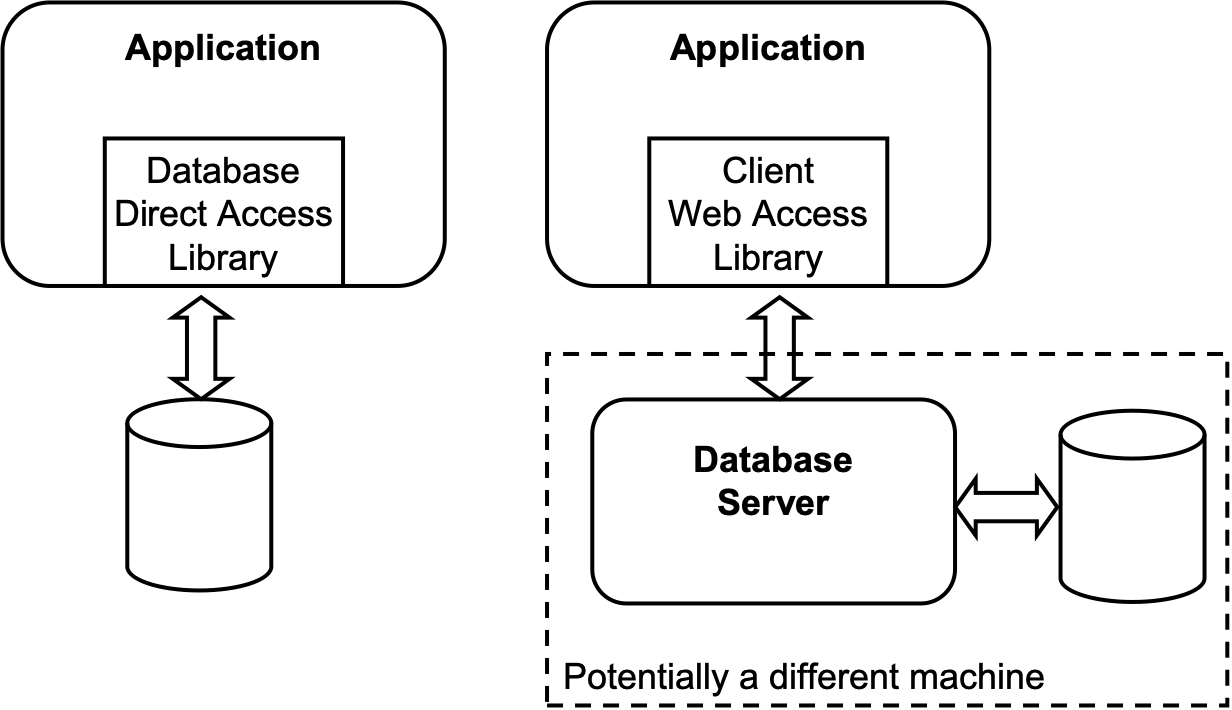
1. Run NetBeans.
2. Switch to the Services Pane.
3. Expand the Databases branch.
4. Right-click on the JavaDB leaf.
5. Select Create Database.
6. Enter the database name consult, and a user APP and a password. Make sure that the password you choose is something you can remember! This way you shouldn't forget what it is. If you don't want that kind of responsibility, use uclan2020.

* **Note:** *Make sure you use the username APP, which is the default database schema. You will have problems if you use another name, because this is used as the schema name. If you get the name wrong, you can create a schema later*.

1. Make a note of the name of the Database Location, the folder where you are storing the database (e.g., C:\documents and settings\dgcampbell.netbeans-derby\consult). [If the IDE complains that the database already exists, navigate to the folder using Windows explorer and delete the consult folder.]. When you click OK, a separate database server application will be started and will create the database for you.
2. When the database has been created, a connection node will appear in the Databases branch.
3. Right-click on the JavaDB leaf and stop the Server.

#### Aside: Using an Embedded Database or a Separate Server

The Derby DBMS can be used as a standalone server or as a library of routines embedded in the application. On a single machine, the embedded server should be slightly less resource hungry (as shown in the left handside of the image below). To convert from one to the other, it is a matter of altering the reference to the database folder: it is a URL for the server and a disk path for the embedded library. To access the server, a client library is needed (derbyclient.jar); the library for the embedded database is in derby.jar. You could change this using the libraries folder in the Projects Tab.



**However, if you are using the embedded database in your application, you can't simultaneously access it from NetBeans, so we will stick with the default, a separate database server.**

### 2. Creating a Table using the NetBeans IDE

Now we have a database configured it is now time to create a table using Netbeans.

1. Right click on the connection node and Choose connect.
2. Once the connection has been made, expand the database and right-click on the Tables node (**This is in the APP folder. Right-click on APP and set APP as the default schema.**) and select **Execute command**.
3. Paste the the SQL code (see, Appendix 1) into the SQL command editor.
4. Execute the SQL you have pasted by pressing Visual Studio User Interface
5. Check that the output page has no errors. If everything is OK, you should see new tables created.

### 3. Generating a database-driven Web application using Netbeans

You are now ready to work through the NetBeans tutorial: Generating a JavaServer Faces 2.0 CRUD Application from a Database (http://netbeans.org/kb/docs/web/jsf20-crud.html, https://netbeans.apache.org/kb/docs/web/jsf20-crud.html), starting from **Examining the Database Structure**, because you have created the database already.

#### Traps to watch for!

1. If you don’t see the Web Application option, look in Java with Ant.
2. Use the Payara server, not Glassfish.
3. Don't forget to change the UNC name for the path to the project folder to N:\ because of problems with Glassfish. If you forget, close the project, and reopen it, replacing the UNC path by N:.
4. There will be a slight difference in the structure because the Tables folder is in a schema folder, APP.

##### Generating entity classes from the database

You will need to scroll to the end of the drop-down list box to find the **New Data Source** entry. The connection will bejdbc:\*\*derby\*\*://localhost:YYYY/consult [APP on APP] not jdbc:\*\*mysql\*\*://localhost:XXXX/consult [root on Default Schema]. Watch for this in the tutorial.

##### Running the application

1. If you have problems, try using Firefox web browser.

##### Populating the Database with an SQL Script

### 4. Answer the following questions...

1. What would you have to change in the program if the database was moved?
2. What sort of database is made up of tables of columns?
3. What language is used in the queries? (See, for example, the named queries in Address.java)
4. What is the advantage of manipulating the database from the IDE at design time?
5. What is the purpose of the Address class?
6. Who wrote the Address class?
7. Explain the meaning of the annotations in the Address class. You may want to compare it with the
8. corresponding database table.
9. What is the purpose of the ClientPK class? Why is there no AddressPK class? (Hint: what is the
10. difference between the primary keys for the two classes, Client and Address)
11. Look at the Web pages in the address folder. Outline how they work.
12. Read the article Introduction to Java Persistence API(JPA) at http://javabeat.net/jpa/ What is a POJO?
13. What is CRUD?
14. What is meant by "a nullable, one-to-many relationship between the CONSULTANT and RECRUITER tables "?
15. What are the advantages and disadvantages of using the JSF framework rather than designing the Web Tier using JSP or Servlets?

## Appendix 1

SQL Command required for "**2. Creating a Table using the NetBeans IDE**". Essential this SQL statement is responsible for creating multiple tables , establishing foreign keys etc.

CREATE TABLE address (  
 address\_id INTEGER NOT NULL GENERATED ALWAYS AS IDENTITY,  
 line1 VARCHAR(50) NOT NULL,  
 line2 VARCHAR(50),  
 city VARCHAR(50) NOT NULL,  
 region VARCHAR(50) NOT NULL,  
 country VARCHAR(50) NOT NULL,  
 postal\_code VARCHAR(50) NOT NULL,  
 CONSTRAINT address\_pk PRIMARY KEY ( address\_id )  
);  
  
CREATE TABLE consultant\_status (  
 status\_id CHAR NOT NULL,  
 description VARCHAR(50) NOT NULL,  
 CONSTRAINT consultant\_status\_pk PRIMARY KEY ( status\_id )  
);  
  
CREATE TABLE consultant (  
 consultant\_id INTEGER NOT NULL GENERATED ALWAYS AS IDENTITY,  
 status\_id CHAR NOT NULL,  
 email VARCHAR(50) NOT NULL,  
 password VARCHAR(50) NOT NULL,  
 hourly\_rate DECIMAL(6,2) NOT NULL,  
 billable\_hourly\_rate DECIMAL(6,2) NOT NULL,  
 hire\_date DATE,  
 recruiter\_id INTEGER,  
 resume LONG VARCHAR,  
 CONSTRAINT consultant\_pk PRIMARY KEY ( consultant\_id )  
);  
  
CREATE TABLE client (  
 client\_name VARCHAR(50) NOT NULL,  
 client\_department\_number SMALLINT NOT NULL,   
 billing\_address INTEGER NOT NULL,  
 contact\_email VARCHAR(50),  
 contact\_password VARCHAR(50),  
 CONSTRAINT client\_pk PRIMARY KEY ( client\_name, client\_department\_number )  
);  
  
CREATE TABLE recruiter (  
 recruiter\_id INTEGER NOT NULL GENERATED ALWAYS AS IDENTITY,  
 email VARCHAR(50) NOT NULL,  
 password VARCHAR(50) NOT NULL,  
 client\_name VARCHAR(50),  
 client\_department\_number SMALLINT,  
 CONSTRAINT recruiter\_pk PRIMARY KEY ( recruiter\_id )  
);  
  
CREATE TABLE project (  
 client\_name VARCHAR(50) NOT NULL,  
 client\_department\_number SMALLINT NOT NULL,  
 project\_name VARCHAR(50) NOT NULL,  
 contact\_email VARCHAR(50),  
 contact\_password VARCHAR(50),  
 CONSTRAINT project\_pk PRIMARY KEY ( client\_name, client\_department\_number, project\_name )  
);  
  
CREATE TABLE project\_consultant (  
 client\_name VARCHAR(50) NOT NULL,  
 client\_department\_number SMALLINT NOT NULL,  
 project\_name VARCHAR(50) NOT NULL,  
 consultant\_id INTEGER NOT NULL,  
 CONSTRAINT project\_consultant\_pk PRIMARY KEY ( client\_name, client\_department\_number, project\_name, consultant\_id )  
);  
  
CREATE TABLE billable (  
 billable\_id BIGINT NOT NULL GENERATED ALWAYS AS IDENTITY,  
 consultant\_id INTEGER NOT NULL,  
 client\_name VARCHAR(50) NOT NULL,  
 client\_department\_number SMALLINT NOT NULL,  
 project\_name VARCHAR(50) NOT NULL,  
 start\_date TIMESTAMP,  
 end\_date TIMESTAMP,  
 hours SMALLINT NOT NULL,  
 hourly\_rate DECIMAL(6,2) NOT NULL,  
 billable\_hourly\_rate DECIMAL(6,2) NOT NULL,  
 description VARCHAR(50),  
 artifacts CLOB,   
 CONSTRAINT billable\_pk PRIMARY KEY ( billable\_id )  
);  
  
ALTER TABLE consultant ADD CONSTRAINT consultant\_fk\_consultant\_status FOREIGN KEY ( status\_id ) REFERENCES consultant\_status ( status\_id );  
ALTER TABLE consultant ADD CONSTRAINT consultant\_fk\_recruiter FOREIGN KEY ( recruiter\_id ) REFERENCES recruiter ( recruiter\_id );  
  
ALTER TABLE client ADD CONSTRAINT client\_fk\_address FOREIGN KEY ( billing\_address ) REFERENCES address ( address\_id );  
ALTER TABLE client ADD CONSTRAINT client\_uk\_billing\_address UNIQUE ( billing\_address );  
  
ALTER TABLE recruiter ADD CONSTRAINT recruiter\_fk\_client FOREIGN KEY ( client\_name, client\_department\_number ) REFERENCES client ( client\_name, client\_department\_number );  
  
ALTER TABLE project ADD CONSTRAINT project\_fk\_client FOREIGN KEY ( client\_name, client\_department\_number ) REFERENCES client ( client\_name, client\_department\_number );  
  
ALTER TABLE project\_consultant ADD CONSTRAINT project\_consultant\_fk\_project FOREIGN KEY ( client\_name, client\_department\_number, project\_name ) REFERENCES project ( client\_name, client\_department\_number, project\_name );  
ALTER TABLE project\_consultant ADD CONSTRAINT project\_consultant\_fk\_consultant FOREIGN KEY ( consultant\_id ) REFERENCES consultant ( consultant\_id );  
  
ALTER TABLE billable ADD CONSTRAINT billable\_fk\_consultant FOREIGN KEY ( consultant\_id ) REFERENCES consultant ( consultant\_id );  
ALTER TABLE billable ADD CONSTRAINT billable\_fk\_project FOREIGN KEY ( client\_name, client\_department\_number, project\_name ) REFERENCES project ( client\_name, client\_department\_number, project\_name );

## Appendix 2

Mutiple SQL statements to Insert data into the database.

SET SCHEMA APP;  
  
INSERT INTO address (line1, line2, city, region, country, postal\_code) VALUES ('100 Data Street', 'Suite 432', 'San Francisco', 'California', 'USA', '94103');  
  
INSERT INTO client (client\_name, client\_department\_number, billing\_address, contact\_email, contact\_password) VALUES ('Big Data Corp.', 2000, 1, 'accounting@bigdatacorp.com', 'accounting');  
  
INSERT INTO project (client\_name, client\_department\_number, project\_name, contact\_email, contact\_password) VALUES ('Big Data Corp.', 2000, 'Secret Project', 'project.manager@bigdatacorp.com', 'project.manager');  
  
INSERT INTO recruiter (email, password, client\_name, client\_department\_number) VALUES ('bob@jsfcrudconsultants.com', 'bob', 'Big Data Corp.', 2000);  
  
INSERT INTO consultant\_status (status\_id, description) VALUES ('A', 'Active');  
  
INSERT INTO consultant (status\_id, email, password, hourly\_rate, billable\_hourly\_rate, hire\_date, recruiter\_id) VALUES ('A', 'janet.smart@jsfcrudconsultants.com', 'janet.smart', 80, 120, '2007-2-15', 1);  
  
INSERT INTO project\_consultant (client\_name, client\_department\_number, project\_name, consultant\_id) VALUES ('Big Data Corp.', 2000, 'Secret Project', 1);  
  
INSERT INTO billable (consultant\_id, client\_name, client\_department\_number, project\_name, start\_date, end\_date, hours, hourly\_rate, billable\_hourly\_rate, description) VALUES (1, 'Big Data Corp.', 2000, 'Secret Project', '2008-10-13 00:00:00.0', '2008-10-17 00:00:00.0', 40, 80, 120, 'begin gathering requirements');  
  
INSERT INTO billable (consultant\_id, client\_name, client\_department\_number, project\_name, start\_date, end\_date, hours, hourly\_rate, billable\_hourly\_rate, description) VALUES (1, 'Big Data Corp.', 2000, 'Secret Project', '2008-10-20 00:00:00.0', '2008-10-24 00:00:00.0', 40, 80, 120, 'finish gathering requirements');