

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.

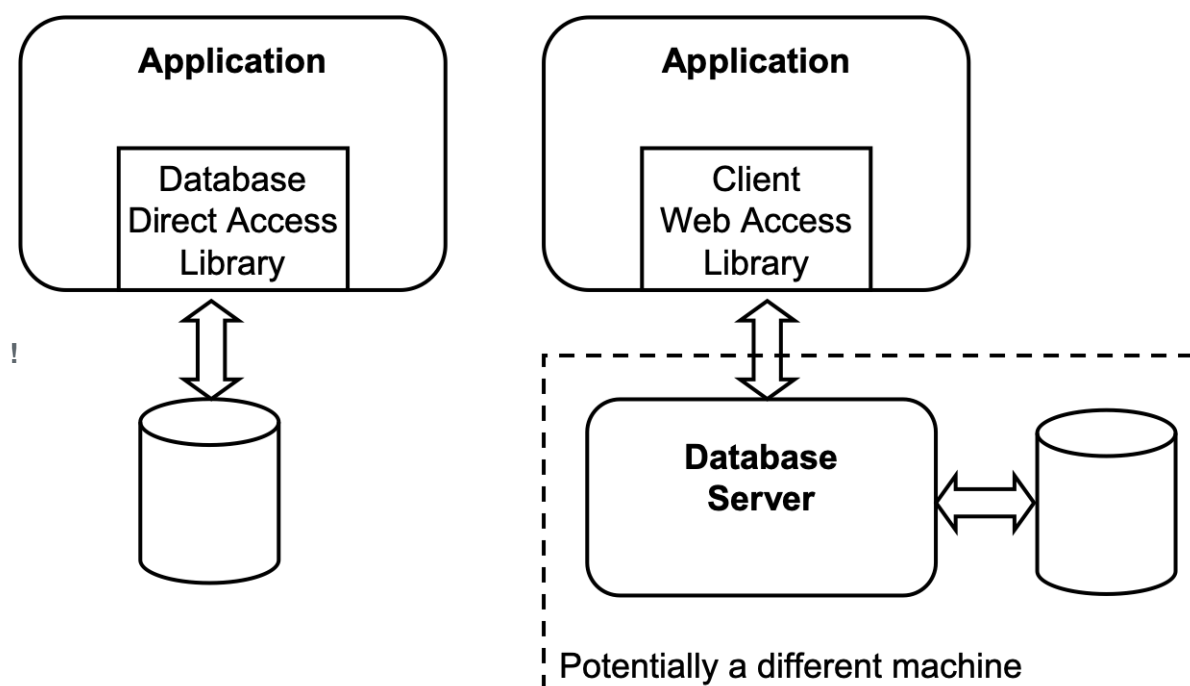
6. Enter the database name `consult`, and a user `APP` and a password. Make sure that the password you choose is something you can remember! This way you shouldn't forget what it is. If you don't want that kind of responsibility, use `uc1an2020`.

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.

7. 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.
8. When the database has been created, a connection node will appear in the Databases branch.
9. 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 
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 be `jdbc:**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.

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');
```