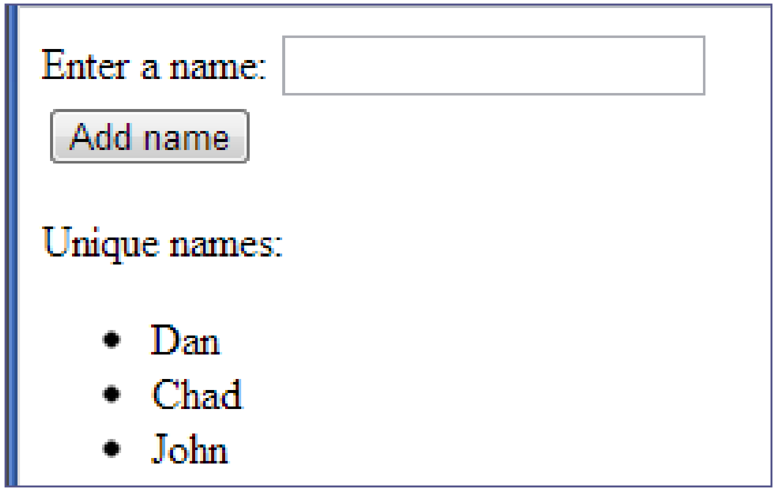
CS 416
Web Programming

Database Connectivity

Dr. Williams
Central Connecticut State University

Review from last class – sample problem


Enter a name:

Unique names:

- Dan
- Chad
- John

- Create a servlet to do the following (use ReviewNames.java as your starting template)
  - Servlet displays page and GETs to itself
  - User is able to submit names
  - At the end of the page output **unique** names that have been entered across all sessions

Database connectivity

- Any non-trivial business application relies on a database
- Dynamic content through retrieving changing data from a database
- Essentially all sophisticated applications both read and write to the database

Database interaction

Two of the key ways of accessing relational data in Java EE are

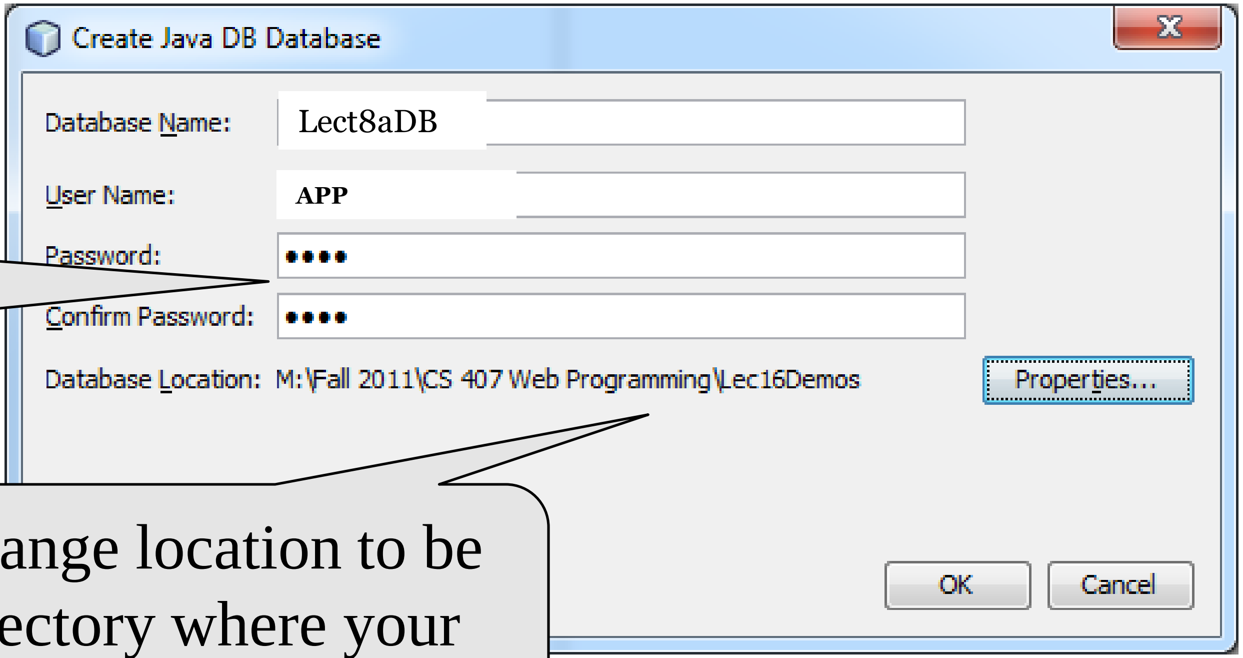
- Java Database Connectivity (JDBC)
  - Standard API for Java applications to interact with a database
  - Although not part of Java EE it is very frequently used in Java EE applications
- Java Persistence API (JPA)
  - Tools for mapping database entries to objects and persisting objects back to the database

Creating a JavaDB instance

In NetBeans

- Go to Window|Services
- Expand Databases
- Right click on Java DB click start server
- Right click Java DB and choose Create database

Create instance



Create Java DB Database

Database Name: Lect8aDB

User Name: APP

Password: ••••

Confirm Password: ••••

Database Location: M:\Fall 2011\CS 407 Web Programming\Lec16Demos

Password: pass

Change location to be directory where your project is

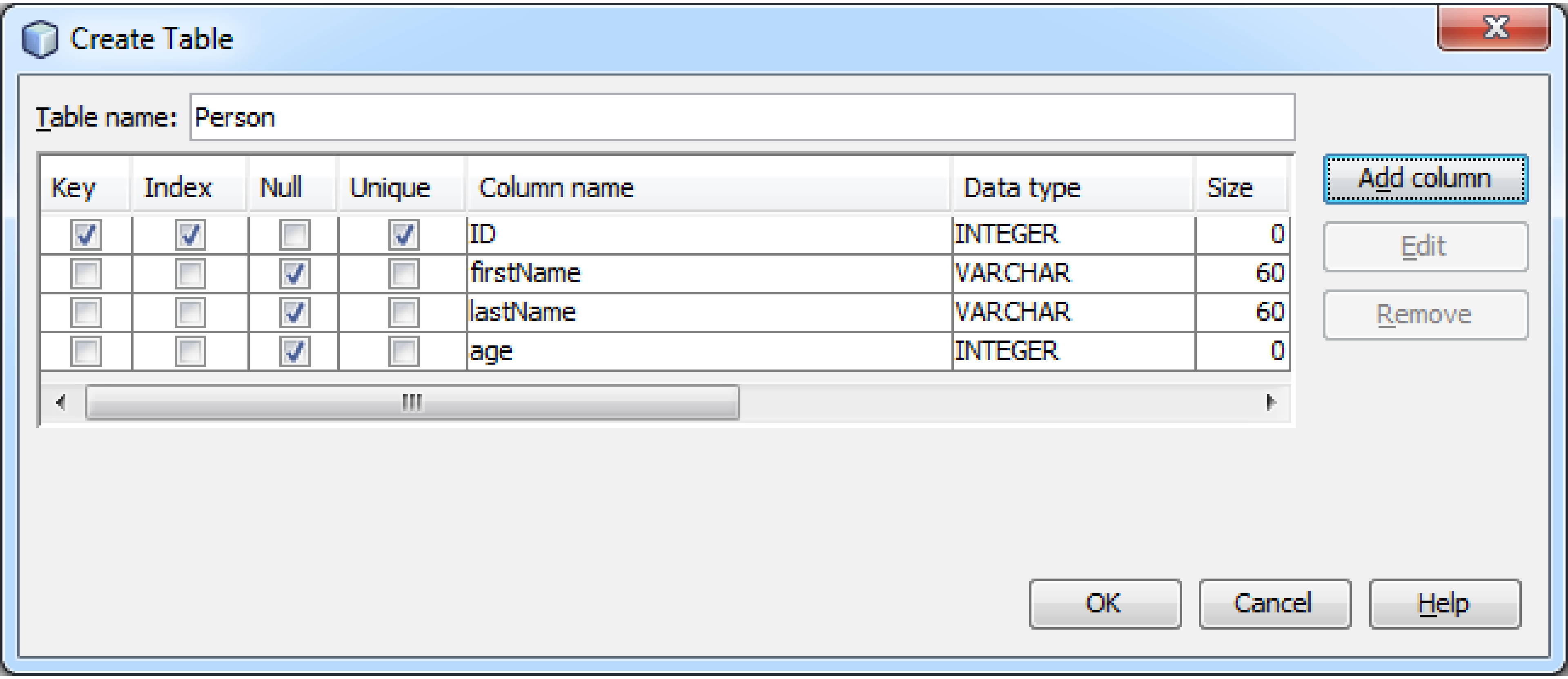
This isn't required, but makes it easier for making sure you submit all of your files to me

Creating schema

- Right click created instance and choose connect
- Expand and right click APP and select Set as default schema
- Click to expand APP

Create PERSON table

- Right click Tables folder and choose create table



Create Table

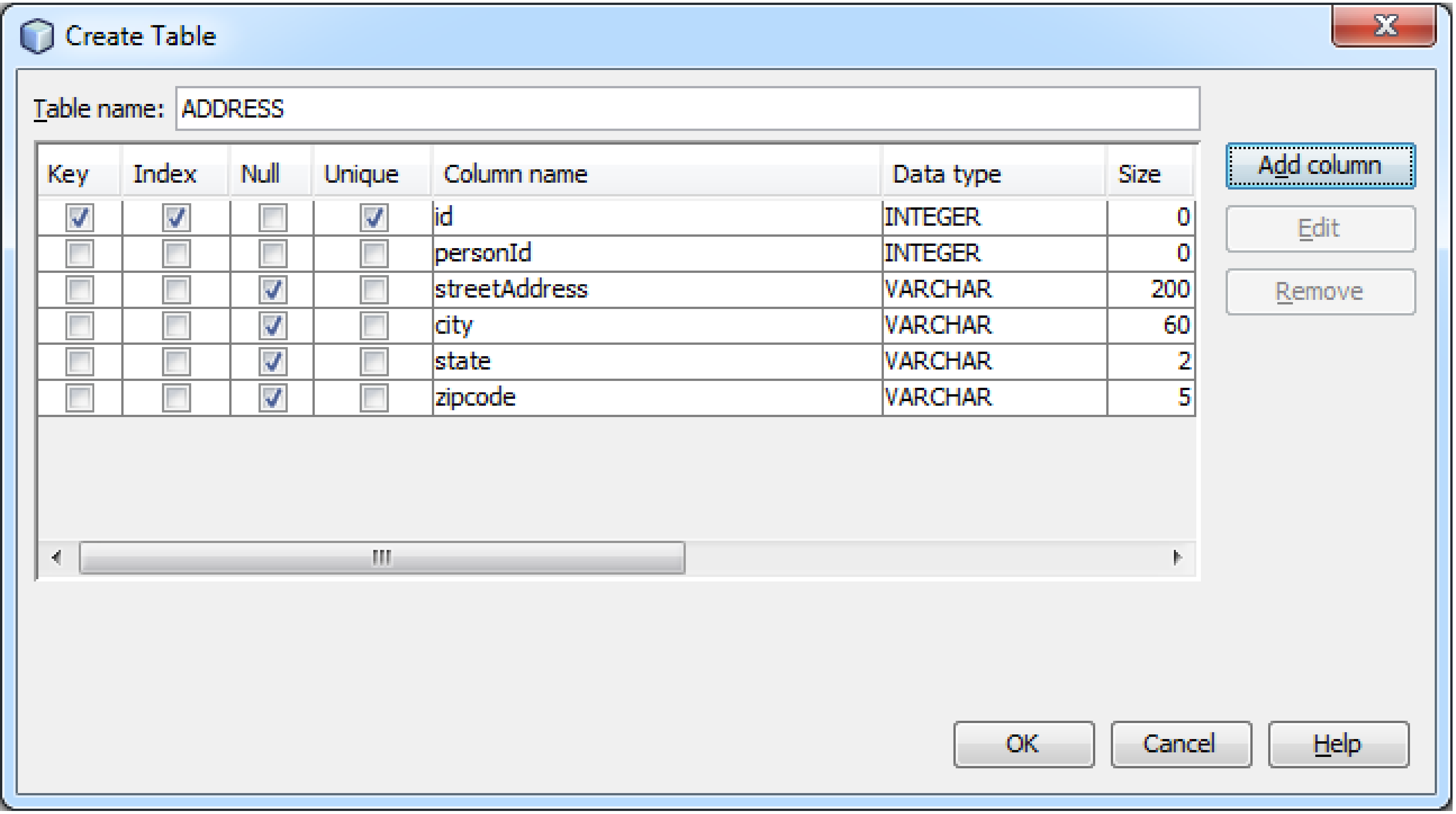
Table name: Person

Key	Index	Null	Unique	Column name	Data type	Size
✓	✓		✓	ID	INTEGER	0
		✓		firstName	VARCHAR	60
		✓		lastName	VARCHAR	60
		✓		age	INTEGER	0

Create data

- Right click person table and choose view data
- Click on insert records icon
- Click on fields within table to add 3 records

Create ADDRESS table



Create Table

Table name: ADDRESS

Key	Index	Null	Unique	Column name	Data type	Size
✓	✓		✓	id	INTEGER	0
				personId	INTEGER	0
		✓		streetAddress	VARCHAR	200
		✓		city	VARCHAR	60
		✓		state	VARCHAR	2
		✓		zipcode	VARCHAR	5

Add foreign key constraint

- Right click tables and select execute command
- This can be used to execute any sql command on the database
- Enter the following to create a foreign key to person

```
ALTER TABLE APP.ADDRESS
ADD FOREIGN KEY (PERSONID)
REFERENCES APP.PERSON(ID)
```

- Add an entry to ADDRESS table

Select SQL cheat sheet

```
select * from PERSON

select id,firstname from PERSON

select * from PERSON where id=?

select * from PERSON where firstname
  like ?
```

? - "Bob**%**“

Join tables:

```
select P.firstname, A.city from
  PERSON P, ADDRESS A where P.ID =
  A.PERSONID
```

Executing queries

- In execute command window write:

```
Select * from person
```

- Execute a select with a join

```
select p.firstname, a.city from
person p  join address a on p.ID
= a.PERSONID
```

Working with our database

- JDBC allows us to connect and execute queries against a database
- Most common way to access data (and best way through JDBC) is through java.sql.PreparedStatement
- Two operations
  - ExecuteQuery – executes a SELECT statement and returns a ResultSet
  - ExecuteUpdate – executes INSERT, UPDATE, DELETE and returns the number of rows affected

(alternative is Statement.execute, but can have bad security implications if you aren't very careful)

Connecting to our database

- Once the database has been created and the schema populated next is registering the database with the application server
  - **Create a connection pool** – allocates a certain number of connections to the database, applications within the application request connections from the pool and return the resource when they are done
  - **Register the Pool in the Java Name Directory Interface (JNDI)** – specify name that can be used to look up the datasource

Creating the connection pool

- Open the Glassfish administration console
  - In NetBeans go to Windows|services
  - In the left pane expand servers
  - Right click GlassFish server and start it if needed then click View Admin Console
- Either select “Create New JDBC Connection Pool” off intro page or click JDBC Connection Pools and then click New

Creating the connection pool cont.

- Give the pool a name: lect8apool
- Change the Resource type to: javax.sql.DataSource
- Change the Database Driver vendor to: JavaDB
- Click Next

Creating connection pool cont.

- In the Pool settings you can configure the size of the pool, how the pool grows, and timeout - for now leave the defaults
- In the additional properties put:

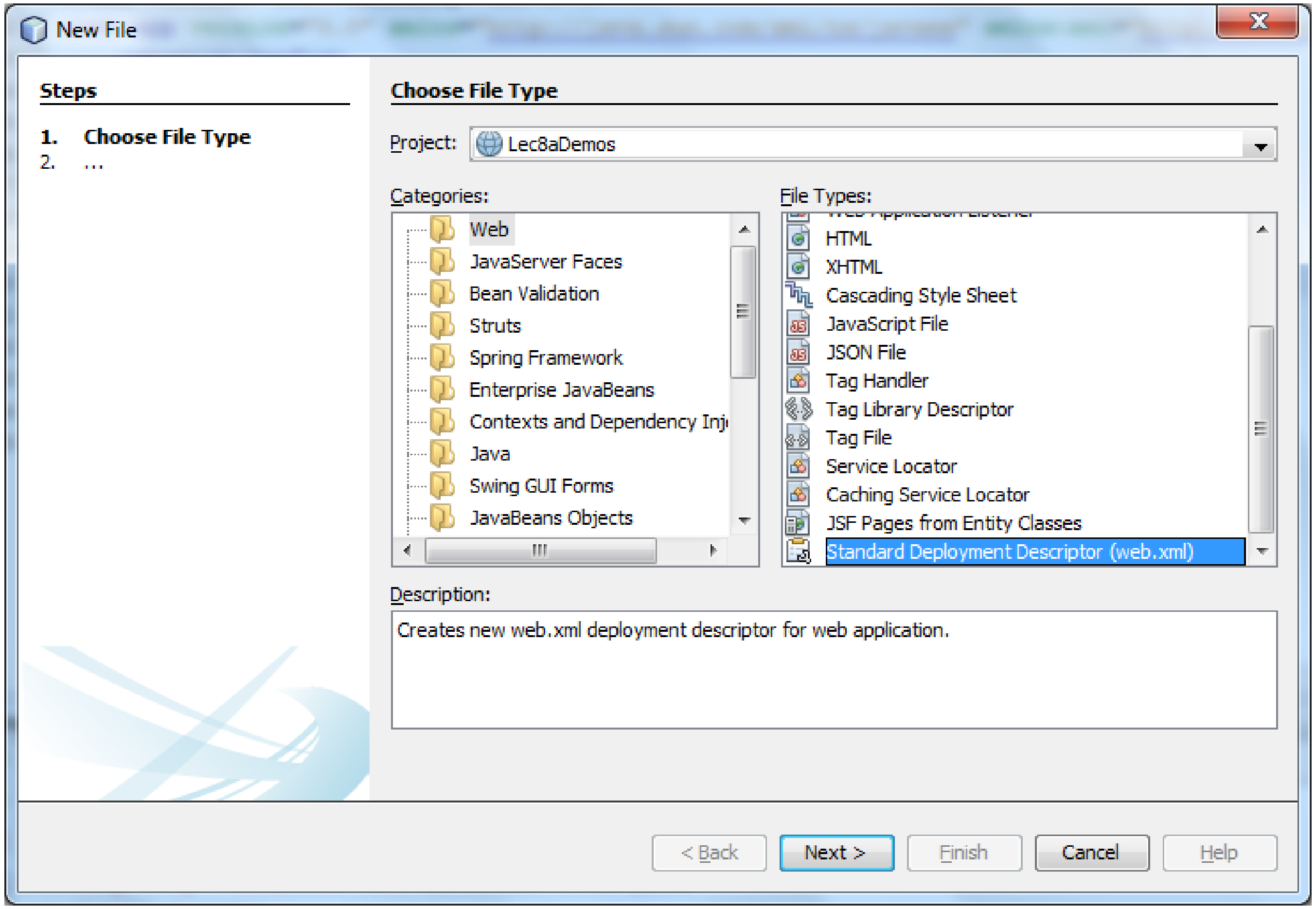
  User: APP
  Password: pass
  DatabaseName: Lect8aDB
  ServerName: localhost
  PortNumber: 1527
- **Delete SecurityMechanism**
- Click Finish
- Test the connection:
  - Click the lect8apool
  - Click ping

Registering data source

- Click the JDBC Resources folder
- Click New
- Enter jdbc/Lect8aDB
- Change the pool name to: lect8apool
- Click OK

Registering resource in project

- Right click on your sources folder and choose New|Other and then under Web pick "Standard deployment descriptor"



Project: Lec8aDemos

Categories:
- Web
- JavaServer Faces
- Bean Validation
- Struts
- Spring Framework
- Enterprise JavaBeans
- Contexts and Dependency Inj
- Java
- Swing GUI Forms
- JavaBeans Objects

File Types:
- HTML
- XHTML
- Cascading Style Sheet
- JavaScript File
- JSON File
- Tag Handler
- Tag Library Descriptor
- Tag File
- Service Locator
- Caching Service Locator
- JSF Pages from Entity Classes
- Standard Deployment Descriptor (web.xml)

Description:

Creates new web.xml deployment descriptor for web application.

Add reference

- Click on “References” across the top, then under Resource References click Add setting the resource name to “jdbc/Lect8aDB”
- **After it’s added click on Source at the top. In the source find the entry:**

  <res-type>javax.sql.DataSource</res-type>
- **Change it to:**

  <res-type>javax.sql.ConnectionPoolDataSource</res-type>

Connecting to our database

- Within servlet add resource of our data source:

```
@Resource(name = "jdbc/Lect8aDB")
private javax.sql.DataSource datasource;
```

- Within processRequest can now access database using:

```
String sql = "select * from PERSON";
Connection connection = datasource.getConnection();
PreparedStatement preparedStatement =
connection.prepareStatement(sql);
ResultSet resultSet = preparedStatement.executeQuery();
while (resultSet.next()) {
  out.println(resultSet.getString("firstName")+"<BR/>");
}
resultSet.close();
preparedStatement.close();
connection.close();
```

(Refer to edu.ccsu.DBTestQuick for example)

Prepared Statements

- SQL statement that is compiled on DB first time it is used, making for very fast execution on subsequent calls
- Also not susceptible to SQL injection attacks
- Two common types of execution
  - executeQuery() - SELECT statement that returns a RecordSet object for working with the results
  - executeUpdate() - Executes INSERT, UPDATE, DELETE and returns an integer, the number of records affected by the update

PreparedStatement parameters

- With prepared statements in addition to having plain SQL you can also specify dynamic parameters by using a question mark “?” as a place holder

```
sql = “select * from person
where firstname like ? and age
= ?
```

- The parameters are then filled in by specifying:

```
preparedStatement.setString(1,”
J%”);
preparedStatement.setInt(2,20);
```

Note index starts at 1 rather than 0

Application flow

- A common application flow is to call a servlet have it retrieve/manipulate data add it to the request/session/context and forward it to another servlet or JSP for display

```
request.getRequestDispatcher("PersonDispl
ay.jsp").forward(request, response);
```

Insert statement cheatsheet

```
insert into PERSON values (?,?,?)
```

Note very problematic as depends on order of database columns if DBA changes these it breaks all of your code or worse

```
insert into PERSON (id,firstname)
 values(?,?)
```

One thing to pay attention to is needing to guarantee primary key is unique – we will cover better ways later, but one option is to get next value

Insert example

- AddPersonServlet
  - Finds next primary key
  - Inserts new record
  - Forwards to allow other servlet to display

Update statement cheatsheet

```
update PERSON set firstname=?,
  lastname=? WHERE id=?

update PERSON set firstname=? WHERE
  lastname=? AND age=?
```

Note like delete, very dangerous if WHERE condition isn’t specific enough

Update example

- UpdatePersonServlet
  - Update first and last name by specified id

Delete statement cheatsheet

```
delete from person where id=?

delete from person where firstname=?
  AND lastname=?
```

Very dangerous if WHERE condition isn’t specific enough

Delete example

- DeletePersonServlet
  - Deletes based on name match (note could delete multiple records)

Input revisited - it’s magic!

- One of the input types we didn’t touch upon earlier was hidden input

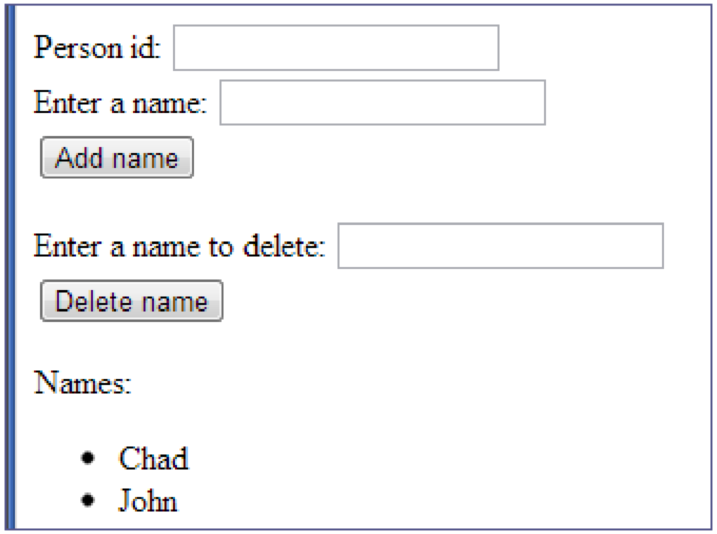
```
<input type="hidden" name="name"
  value="Chad"/>
```

- Form element doesn’t show up on page but value of element gets submitted with rest of form
- Why do I care?
  - Very useful for keeping track of identifier (or other info) without having to show it to the user!

Display names revisited

- Revise DisplayNamesServlet to display names with submit button to direct to page to display that name by the id passed **use GET**
- Create DisplayPersonServlet to display a name with the passed id with the first and last name in textboxes (will become apparent why later)
- Take advantage of GET form using hyperlinks

Review problem revisited (use ReviewRevisited.java as your template)


Person id:

Enter a name:

Add name

Enter a name to delete:

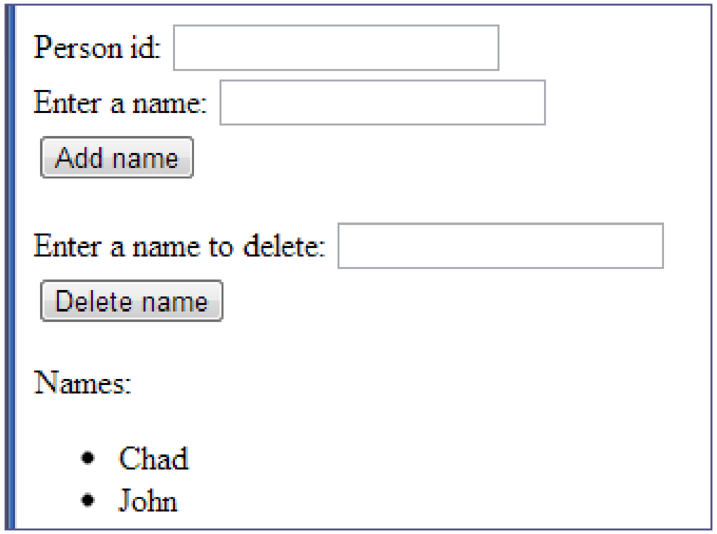
Delete name

Names:

- Chad
- John

- Create a servlet modeled off the review problem
- When the person enters a name it should be added to the person table
- Display should be done by reading from the database
- Do not worry about whether the names are unique
- If you have time, add functionality allowing you to delete a name

Review problem revisited (use ReviewRevisited.java as your template)


Person id:

Enter a name:

Enter a name to delete:

Names:

- Chad
- John

- Create a servlet modeled off the review problem
- When the person enters a name it should be added to the person table
- Display should be done by reading from the database
- Do not worry about whether the names are unique
- If you have time, add functionality allowing you to delete a name