

JSF, Entitys, Managed Beans and Database

At This Point

- So far we have concentrated mainly in implementing the UI, so you should be familiar with the basic concepts how you create UI's and inject data to UI using Managed Beans.
- Next thing to focus is how you use database from your JFS application and how you update and insert data from database and present the data in UI.

Creating Database

- First we need to create a database.
- We use dbForge Express database designer tool for this. You can download it from here:[dbForge](#)
- After downloading just double click the .exe file and follow instructions.

Configuring dbForge

- When you start dbForge for the first time, it will ask you the connection attributes.
- Just fill up the next information into the fields and press Test Connection.
- NOTE! The password is also “root”

Configuring dbForge

The screenshot shows the 'Database Connection Properties' dialog box with the 'General' tab selected. The dialog is titled 'Database Connection Properties' and has a close button (X) in the top right corner. It contains three tabs: 'General', 'Advanced', and 'Embedded'. The 'General' tab is active, showing fields for 'Host', 'Port', 'User', 'Password', 'Database', and 'Connection name'. The 'Host' field is set to 'localhost', 'Port' to '3306', 'User' to 'root', and 'Password' to '.....'. The 'Database' field is empty. There are two checkboxes: 'Allow saving password' (checked) and 'Show all databases' (checked). The 'Connection name' field is set to 'localhost1'. On the right side of the dialog, there are three buttons: 'OK', 'Cancel', and 'Help'. At the bottom center, there is a 'Test Connection' button.

Database Connection Properties

General Advanced Embedded

Specify following to connect to MySQL server:

Host: localhost

Port: 3306

User: root

Password:

☒ Allow saving password

Database:

☒ Show all databases

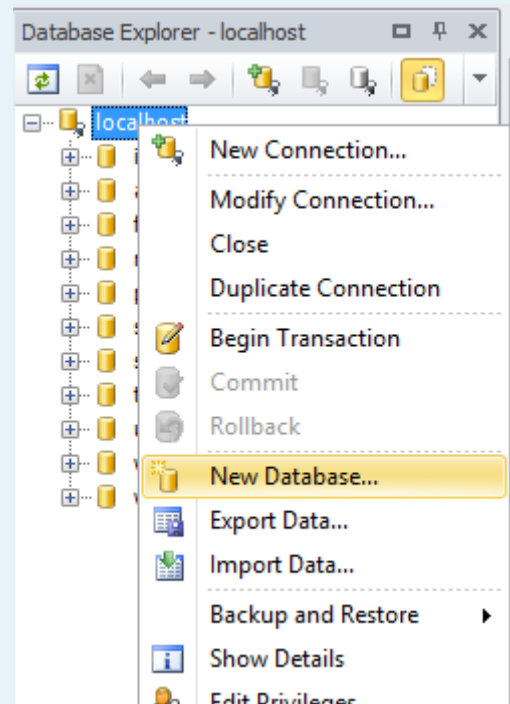
Connection name:
localhost1

OK
Cancel
Help

Test Connection

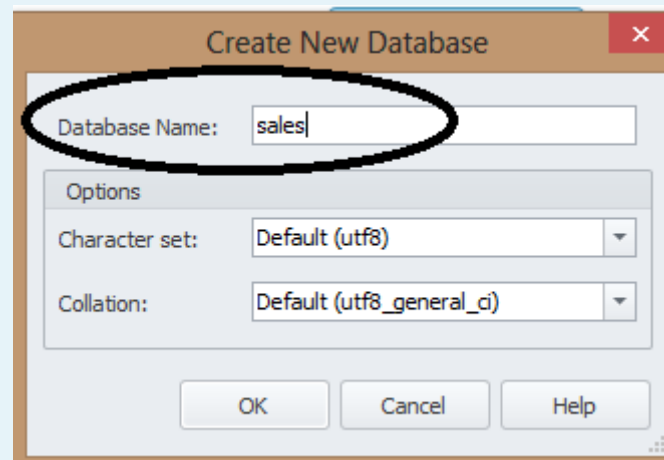
Creating a Database

- Click mouse right over “localhost” in Database explorer and select → New Database



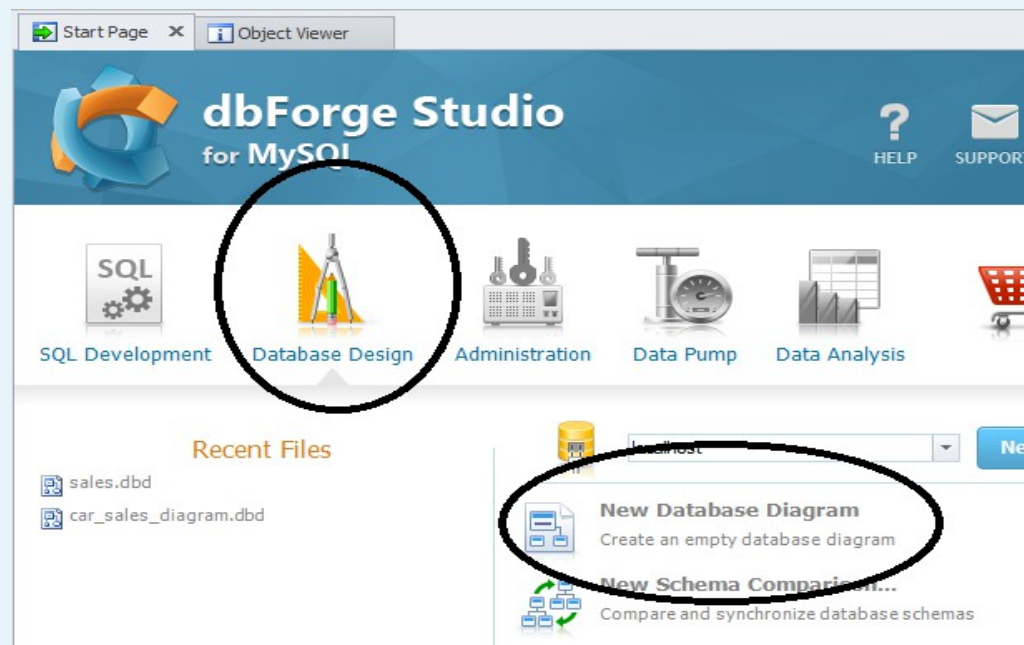
Creating a Database

- Give some name for the database and press Ok button.



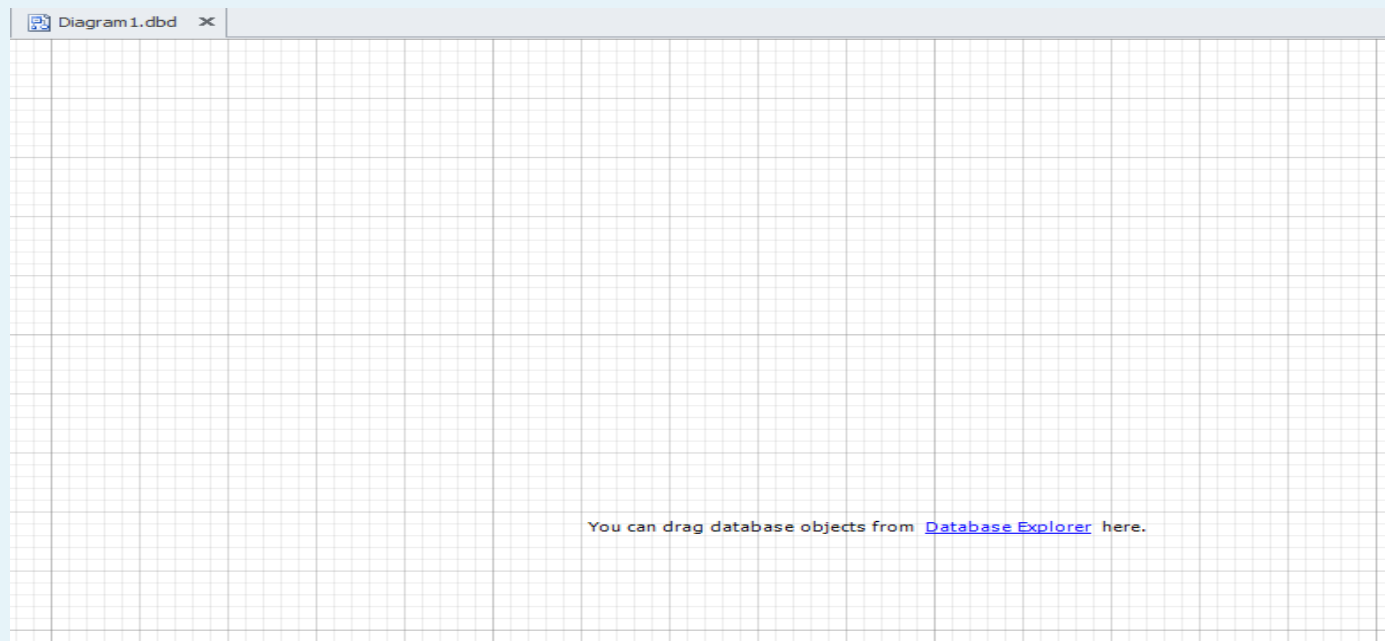
Creating a Database

- Next from dbForge Studio select Database Design and click New Database Diagram



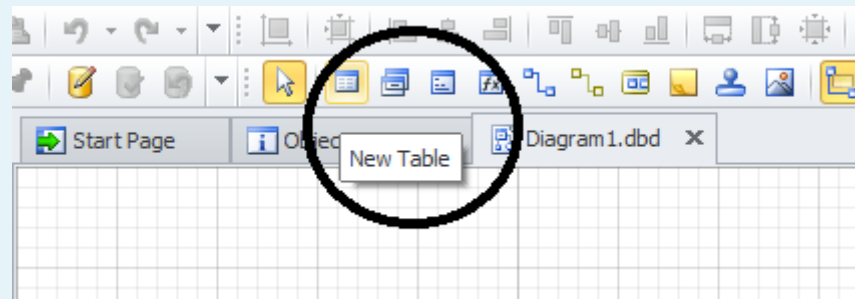
Creating a Database

- You should now have an empty diagram open where you can now start to design database tables...



Creating a Table

- On the toolbar click New Table icon



Creating a Table

- When the new table icon is selected you can now click on design area and next kind of editor should be open

table1

Main Constraints Indexes Options SQL Data

Name: table1 Database:
 Comment:
 Engine: INNODB

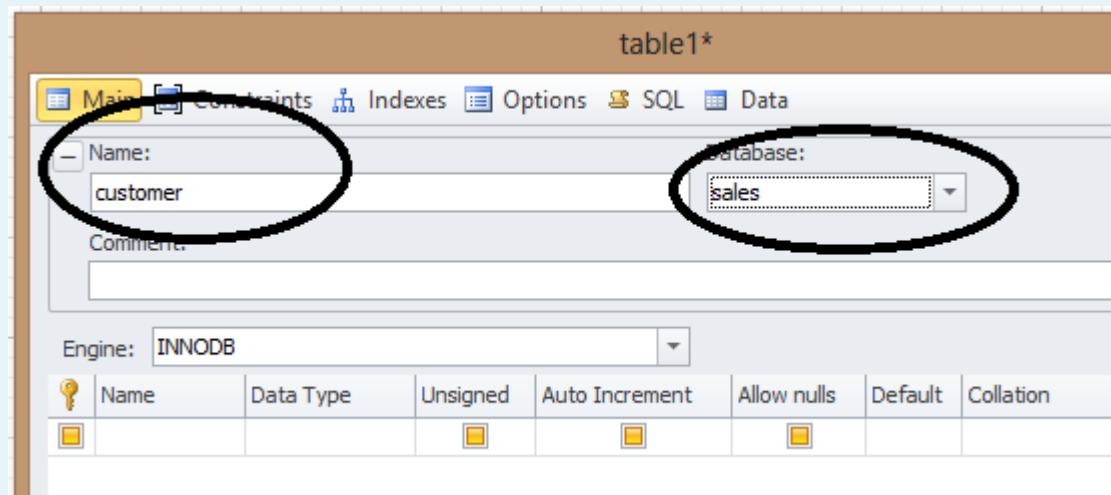
	Name	Data Type	Unsigned	Auto Increment	Allow nulls	Default	Collation	Co

Update Database Script Changes (to Clipboard)

Reset OK Cancel Help

Creating a Table

- Next give a name for the table and select a database where the table should be appended (if you named it sales select that from the list)



The screenshot shows a dialog box for creating a new table, titled 'table1*'. It has several tabs: 'Main', 'Constraints', 'Indexes', 'Options', 'SQL', and 'Data'. The 'Main' tab is selected. In the 'Name' field, the text 'customer' is entered. In the 'Database' dropdown menu, 'sales' is selected. Both the 'Name' field and the 'Database' dropdown are circled in black. Below these fields is a 'Comment' text area. At the bottom, the 'Engine' is set to 'INNODB'. Below the engine selection is a table with columns: 'Name', 'Data Type', 'Unsigned', 'Auto Increment', 'Allow nulls', 'Default', and 'Collation'. The first row of this table has checkboxes for 'Unsigned', 'Auto Increment', and 'Allow nulls'.

Name	Data Type	Unsigned	Auto Increment	Allow nulls	Default	Collation
		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		

Creating a Table

- Next we define database rows. After you have defined the rows, just press ok and the table should be now in design area.

table1*

Main Constraints Indexes Options SQL Data

Name: customer Database: sales

Comment:



Engine: INNODB

	Name	Data Type	Unsigned	Auto Increment	Allow nulls	Default	Collation
<input checked="" type="checkbox"/>	customer_id	INT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>		
<input type="checkbox"/>	customer_name	VARCHAR(50)	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>		
<input type="checkbox"/>	customer_address	VARCHAR(255)	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>		
<input type="checkbox"/>			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		

customer	
customer_id	INT
customer_name	VARCH...
customer_address	VARCH...
Constraints	

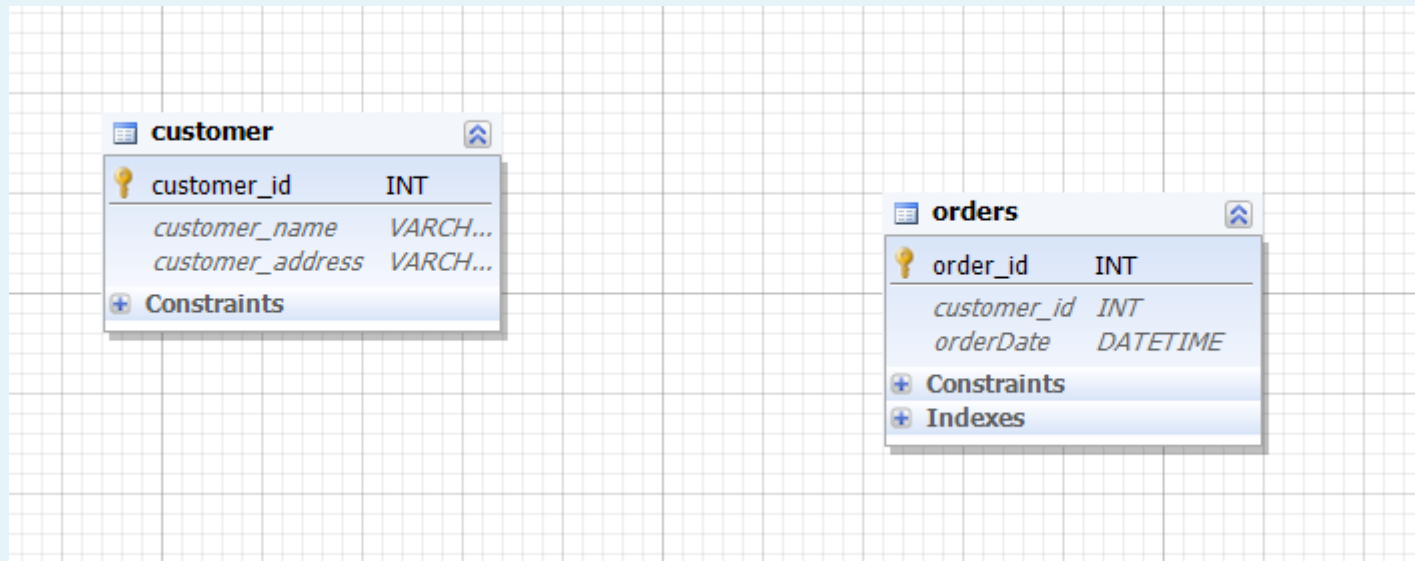
Creating Relations

- Next we want to create a orders table and make a relation to customer table. This relation should be one to many since one customer can have several orders.
- So create a new table called orders with next information and press ok.

	Name	Data Type	Unsigned	Auto Increment	Allow nulls	Default	Colla
<input checked="" type="checkbox"/>	 order_id	INT(11)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>		
<input type="checkbox"/>	customer_id	INT(11)	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	NULL	
<input type="checkbox"/>	orderDate	DATETIME	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	NULL	
<input type="checkbox"/>			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		

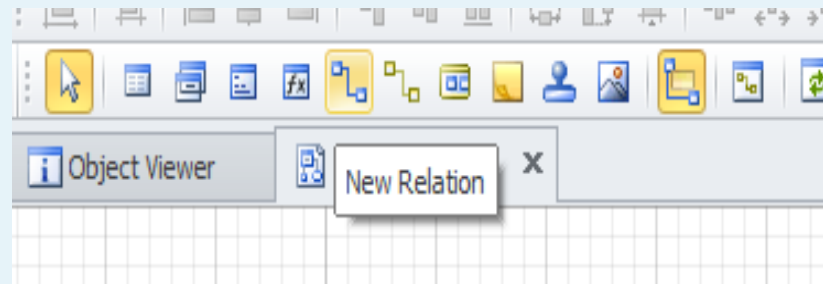
Creating Relations

- Now the situation should be like this:



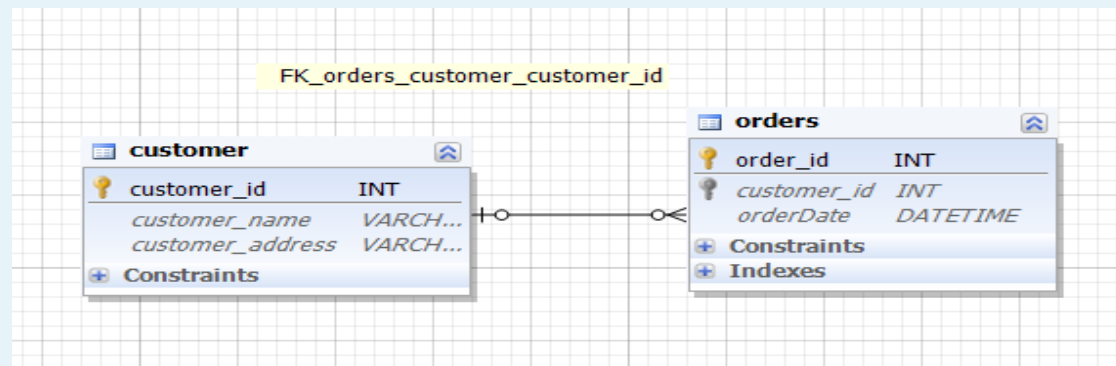
Creating Relations

- Next select New Relation icon from toolbar



Creating Relations

- No when new relation icon is selected, click order_id row in orders table and drag mouse over customer table order_id row. Release now mouse. Press Ok in dialog which opens. Now you should have one to many relation between tables



Exercise

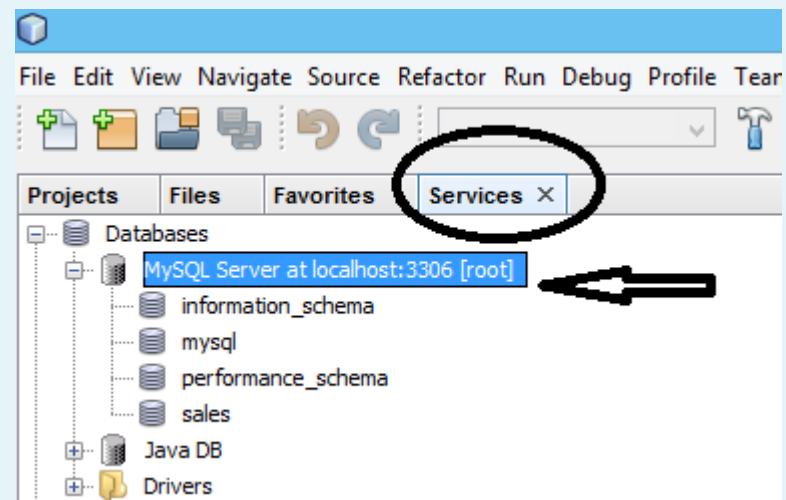
- Now append a new table in our database called product. Insert next rows to that
 - product_id which is a primary key and autoincrement
 - order_id type is int
 - product_name type varchar
 - product_price type is float
- Make one to many relation between orders table and product table.

Integrating DB to JSF project

- Next thing to do is integrate the previously created database in to our JSF application.
- This can be done in several way
 - Use JPA and Entities
 - Use Hibernate
 - Use directly Java JDBC
- We follow the first option in this case.

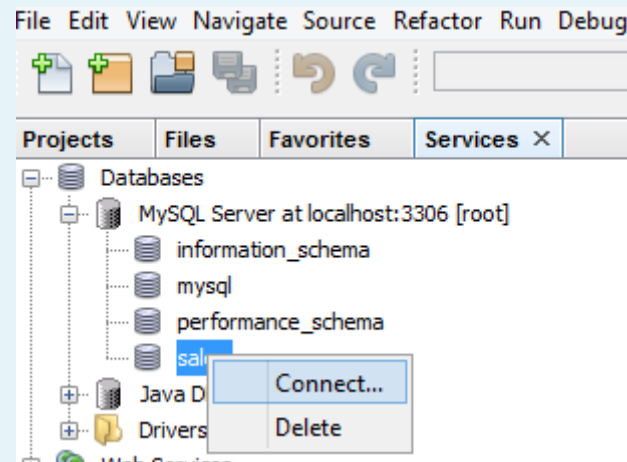
Integrating DB to JSF project

- Now open NetBeans IDE and create a new JSF project.
- Then open the service tab in project explorer. Make sure that you are connected to database server



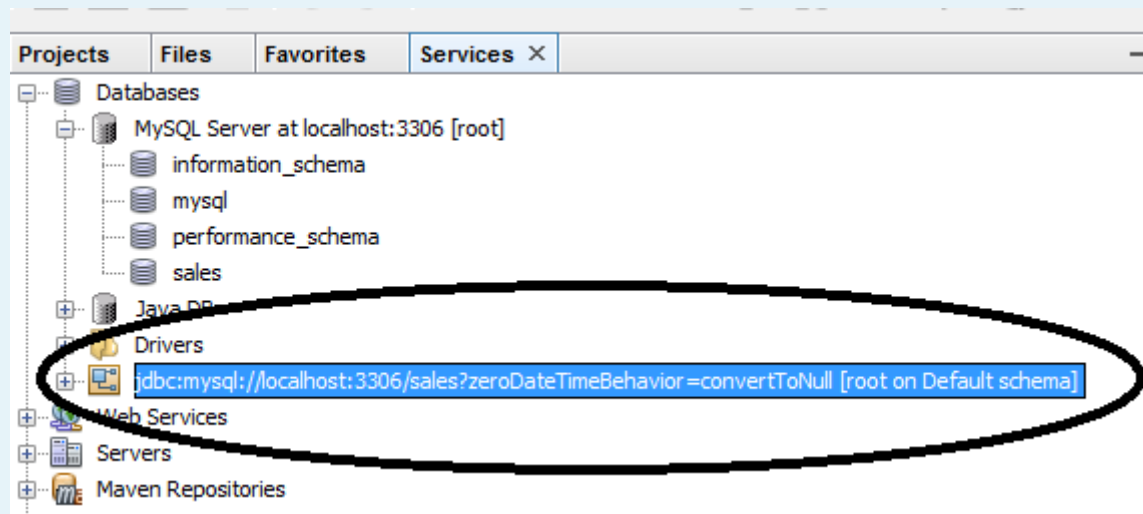
Integrating DB to JSF project

- Right mouse click over the sales database in Databases and select connect...



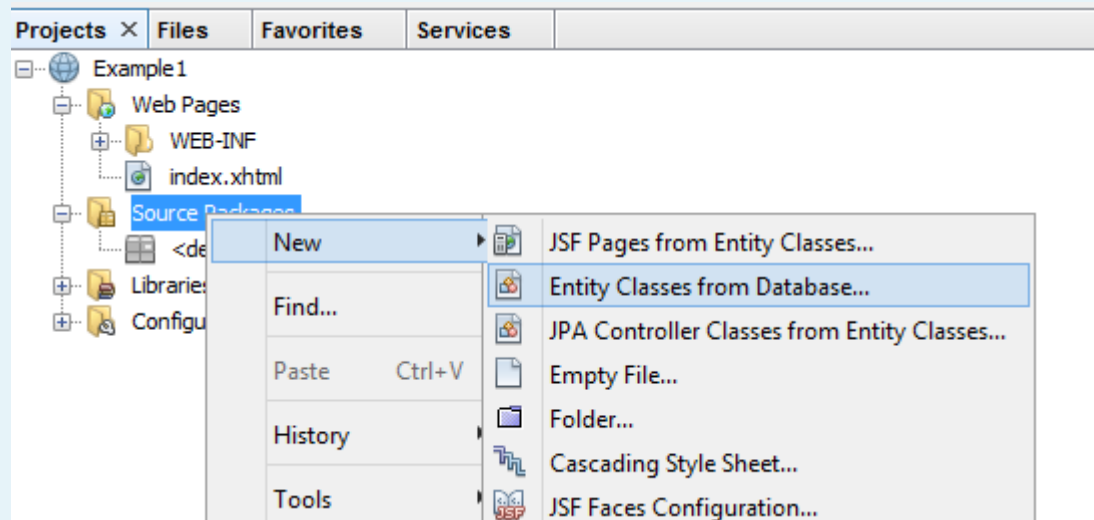
Integrating DB to JSF project

- Now you should see that you are connected to sales database...



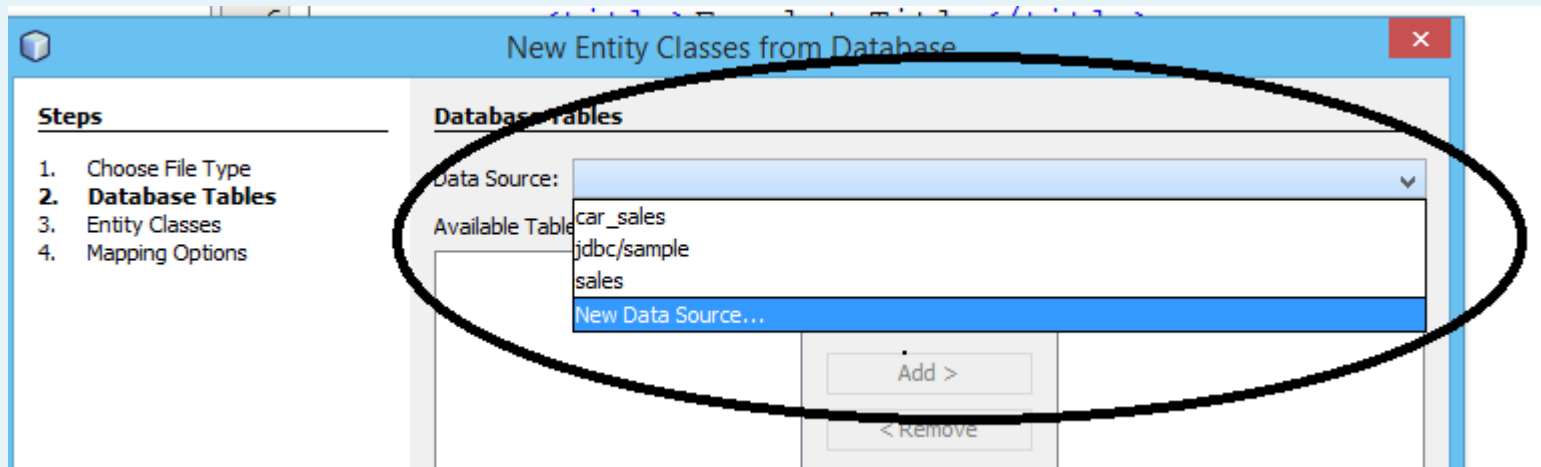
Creating Entitys

- Now open the Projects tab again. Right mouse click over Source Packages folder. Select New-> Entity classes from database



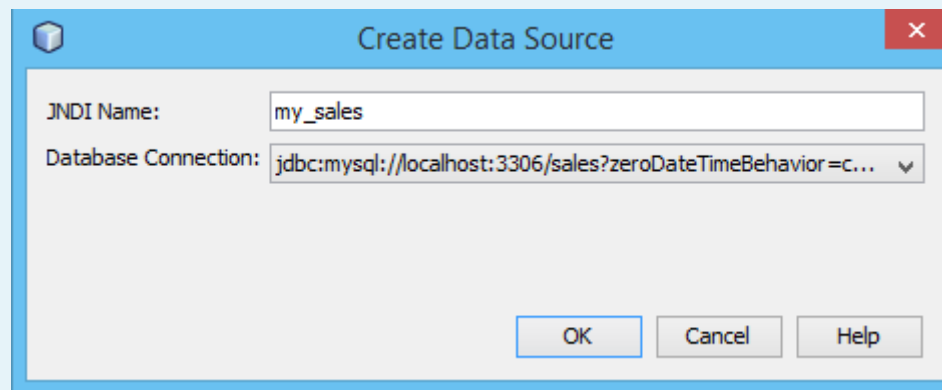
Creating Entitys

- From Data Source select New Data Source



Creating Entitys

- Give some JNDI name and select the sales database connection that we previously created and press ok button.

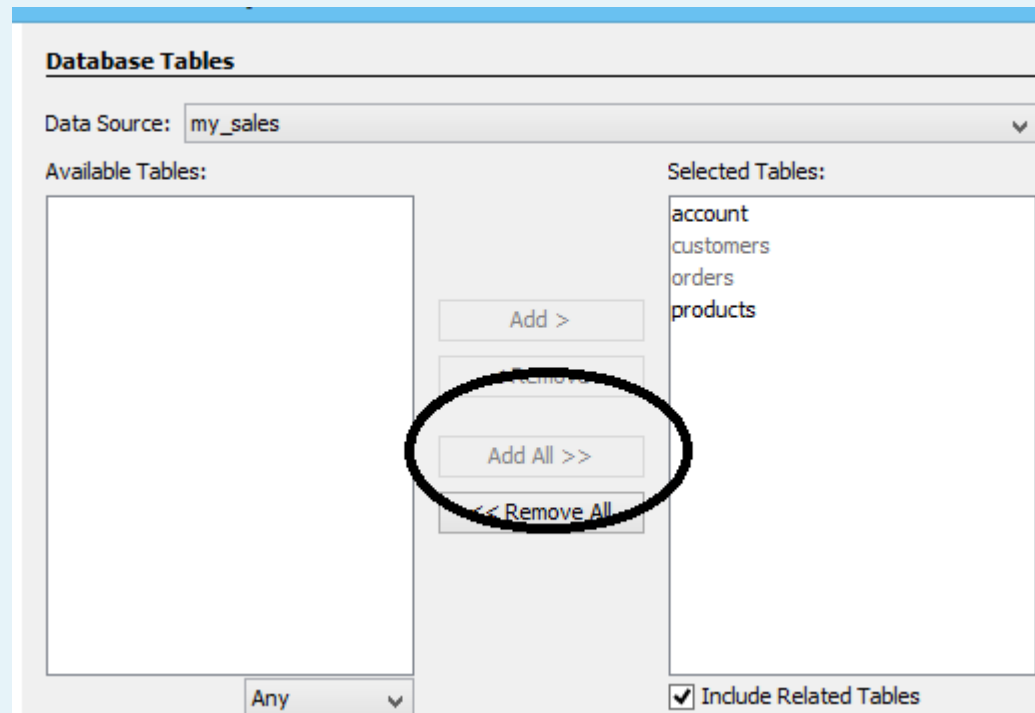


The image shows a 'Create Data Source' dialog box with a blue title bar and a red close button. It contains two input fields: 'JNDI Name:' with the text 'my_sales' and 'Database Connection:' with a dropdown menu showing 'jdbc:mysql://localhost:3306/sales?zeroDateTimeBehavior=c...'. At the bottom are three buttons: 'OK', 'Cancel', and 'Help'.

Field	Value
JNDI Name:	my_sales
Database Connection:	jdbc:mysql://localhost:3306/sales?zeroDateTimeBehavior=c...

Creating Entitys

- Next press Add all button and press next button



Creating Entitys

- Give some package name for your entity classes and press finish.

The screenshot shows the 'New Entity Classes from Database' dialog box. The 'Entity Classes' section is active, showing a table with columns 'Database Table', 'Class Name', and 'Generation Type'. The table contains three rows: 'account' (Account, New), 'customers' (Customers, New), and 'orders' (Orders, New). Below the table, the 'Project' field is set to 'Example1', the 'Location' is 'Source Packages', and the 'Package' field is 'com.opiframe.entity', which is circled in black. At the bottom, there are three checked options: 'Generate Named Query Annotations for Persistent Fields', 'Generate JAXB Annotations', and 'Create Persistence Unit'. The bottom buttons are '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'.

Database Table	Class Name	Generation Type
account	Account	New
customers	Customers	New
orders	Orders	New

Project: Example1

Location: Source Packages

Package: com.opiframe.entity

☒ Generate Named Query Annotations for Persistent Fields

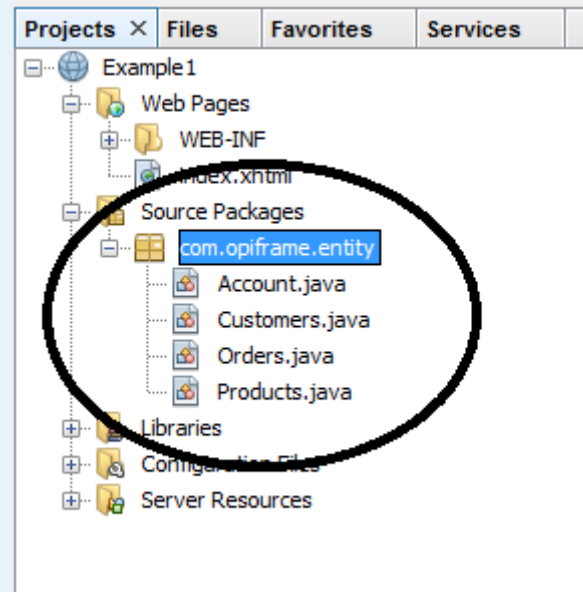
☒ Generate JAXB Annotations

☒ Create Persistence Unit

< Back Next > Finish Cancel Help

Creating Entitys

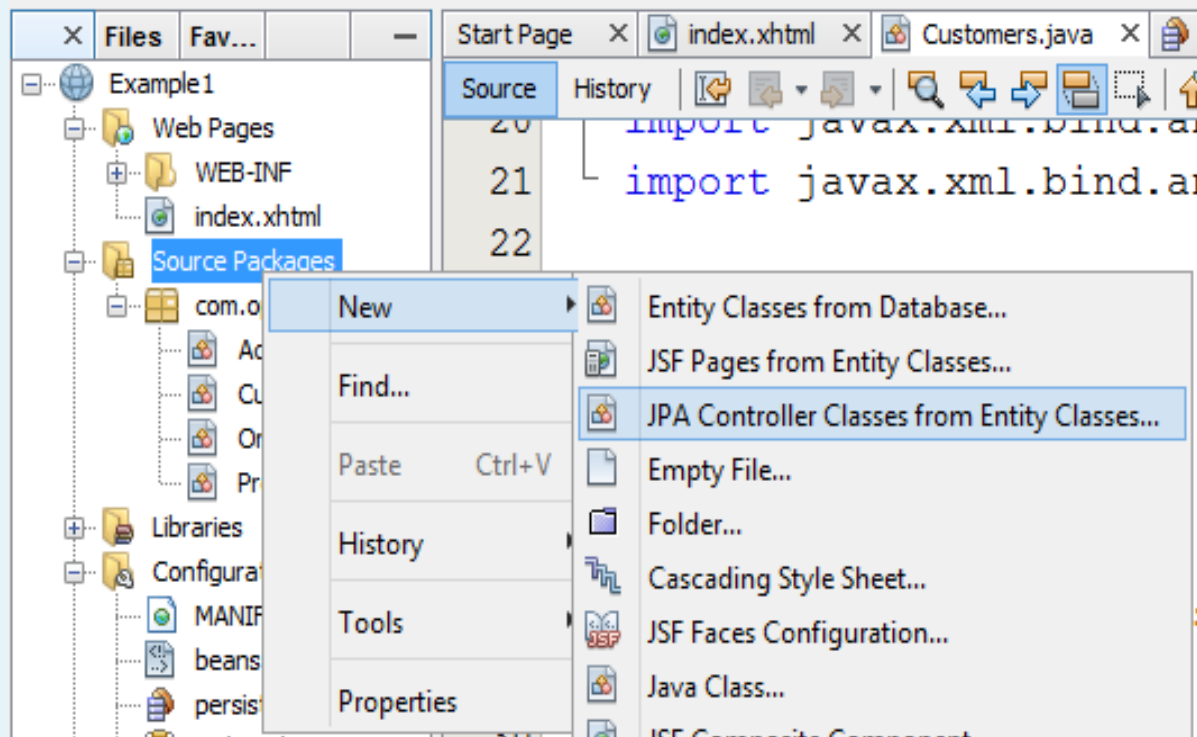
- Now you should have all the entity classes in the package name you defined...



Creating JPA Controller

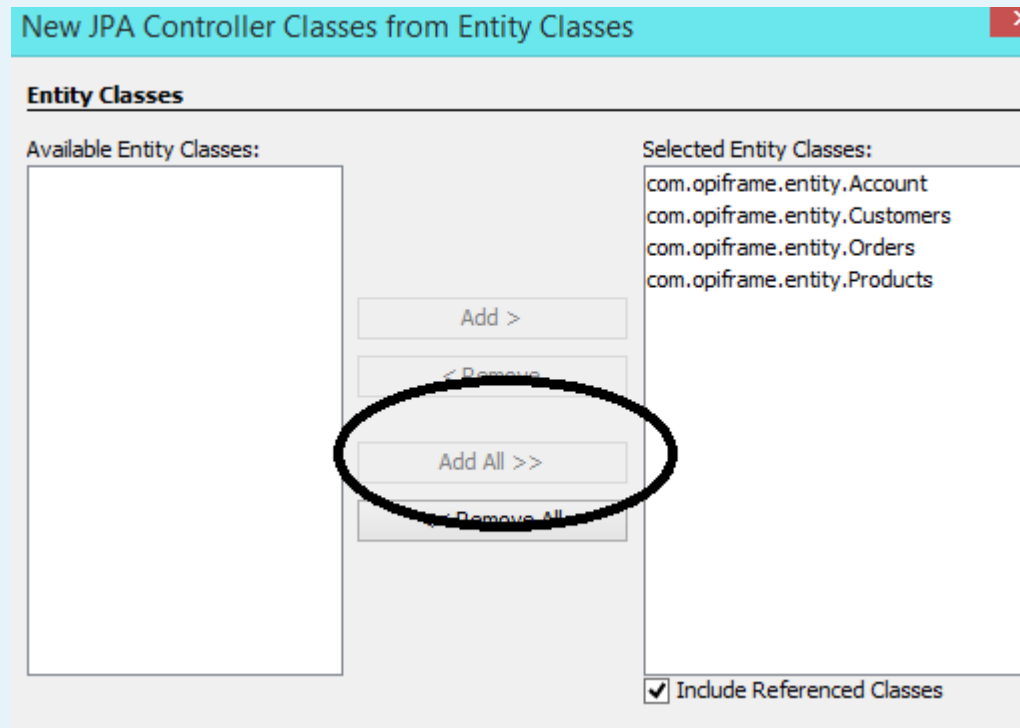
- Next thing to do is to create an JPA controller that actually connects to database and contains all the basic operations so that we can create, delete, update and read from database.
- Click right mouse button over Souorce Packages folder. Select New->JPA Controller Classes from Entity Classes

Creating JPA Controller



Creating JPA Controller

- From opened window press Add All Button and then next...



Creating JPA Controller

- Define the package name and press Finish button...

New JPA Controller Classes from Entity Classes

Generate JPA Controller Classes

Specify the location of the JPA controller classes and related classes.

Project: Example1

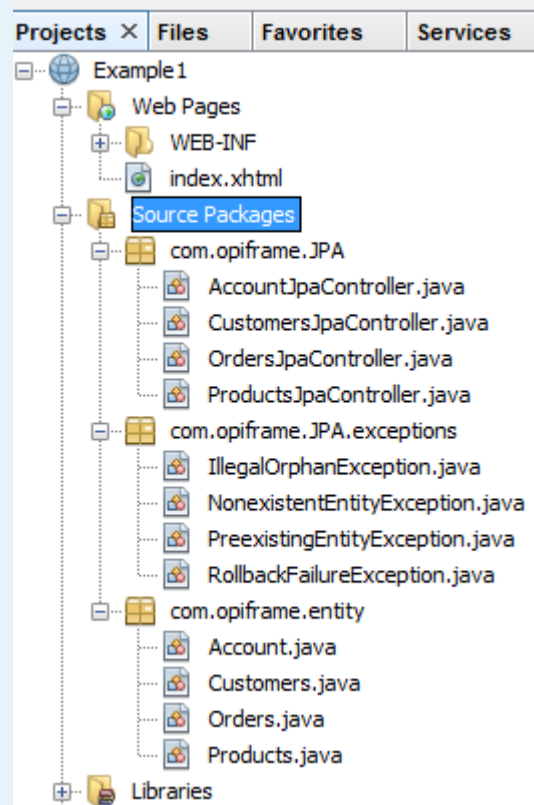
Location: Source Packages

Package: com.opiframe.JPA

< Back Next > Finish

Creating JPA Controller

- Now you should have following project structure....



JPA Controller

- Now if you open one of the JPA Controllers i.e. CustomersJpaController.java file you can find the CRUD methods there for customer table operations....

```
public void create(Products products) throws RollbackFailureException,
    EntityManager em = null;
    try {
        utx.begin();
        em = getEntityManager();
        Orders orderId = products.getOrderId();
        if (orderId != null) {
```

JPQL Language

- Now if you open a Customers.java file you can see basic JPQL queries for database table customers

```
@Entity
@Table(name = "customers")
@XmlRootElement
@NamedQueries({
    @NamedQuery(name = "Customers.findAll", query = "SELECT c FROM Customers c"),
    @NamedQuery(name = "Customers.findById", query = "SELECT c FROM Customers c WHERE c.customerId = :id"),
    @NamedQuery(name = "Customers.findByName", query = "SELECT c FROM Customers c WHERE c.name = :name"),
    @NamedQuery(name = "Customers.findByAddress", query = "SELECT c FROM Customers c WHERE c.address = :address"),
    @NamedQuery(name = "Customers.findByEmail", query = "SELECT c FROM Customers c WHERE c.email = :email"),
    @NamedQuery(name = "Customers.findByPhone", query = "SELECT c FROM Customers c WHERE c.phone = :phone")})
```

JPQL Language

- JPQL a.k.a Java Persistence Query Language is language you can use to make queries to database from your application.
- Next slides contains a short tutorial about JPQL language.
- You need to know basic rules of JPQL to make your own custom queries for database.

Query Language Terminology

- **Abstract schema:** The persistent schema abstraction (persistent entities, their state, and their relationships) over which queries operate. The query language translates queries over this persistent schema abstraction into queries that are executed over the database schema to which entities are mapped.

Query Language Terminology

- **Abstract schema type:** The type to which the persistent property of an entity evaluates in the abstract schema. That is, each persistent field or property in an entity has a corresponding state field of the same type in the abstract schema. The abstract schema type of an entity is derived from the entity class and the metadata information provided by Java language annotations.

Query Language Terminology

- **Backus-Naur Form (BNF):** A notation that describes the syntax of high-level languages. The syntax diagrams in this slide set are in BNF notation.

Query Language Terminology

- **Navigation:** The traversal of relationships in a query language expression. The navigation operator is a period.
- **Path expression:** An expression that navigates to a entity's state or relationship field.
- **State field:** A persistent field of an entity.
- **Relationship field:** A persistent field of an entity whose type is the abstract schema type of the related entity.

Creating Queries Using the JPQL

- The `EntityManager.createQuery` and `EntityManager.createNamedQuery` methods are used to query the datastore by using Java Persistence query language queries.

Creating Queries Using the JPQL

- The `createQuery` method is used to create dynamic queries, which are queries defined directly within an application's business logic:

```
public List findWithName(String name) {  
    return em.createQuery(  
        "SELECT c FROM Customer c WHERE c.name LIKE :custName")  
        .setParameter("custName", name)  
        .setMaxResults(10)  
        .getResultList();  
}
```

Creating Queries Using the JPQL

- The `createNamedQuery` method is used to create static queries, or queries that are defined in metadata by using the `javax.persistence.NamedQuery` annotation.
- The `name` element of `@NamedQuery` specifies the name of the query that will be used with the `createNamedQuery` method.
- The `query` element of `@NamedQuery` is the query

Creating Queries Using the JPQL

```
@NamedQuery(  
    name="findAllCustomersWithName",  
    query="SELECT c FROM Customer c WHERE c.name LIKE :custName"  
)
```

Here's an example of `createNamedQuery`, which uses the `@NamedQuery`:

```
@PersistenceContext  
public EntityManager em;  
...  
customers = em.createNamedQuery("findAllCustomersWithName")  
    .setParameter("custName", "Smith")  
    .getResultList();
```

JPQL

- For more information about JPQL syntax and usage see: [JPQL Doc link](#)