T.P. 3
Base de données, JDBC et Mapping Objet-relationnel

L'objectif de ce TP est de vous faire pratiquer l'API JDBC qui permet d'accéder à une base de données relationnel à partir d'un composant Java J2EE. Dans la première partie du TP, une base de données Derby sera créée à l'aide d'Eclipse.
L'objectif de la deuxième partie est de mettre en place un site web pour l'administration et la commande de pizzas :

- une pizza est déterminée par son type et son prix à l'unité.
- un stock (quantité) est donné pour chaque type de pizza
- une commande se définit par un type de pizza, une quantité, le total de la commande, l'email de confirmation de la commande.

1. Base de données

Eclipse contient par défaut la base de données Derby développée par Sun. Derby permet de stocker les tables de la base de données directement dans le système de fichiers (sous forme d'un dossier) sans passer par la configuration du serveur de base de données.

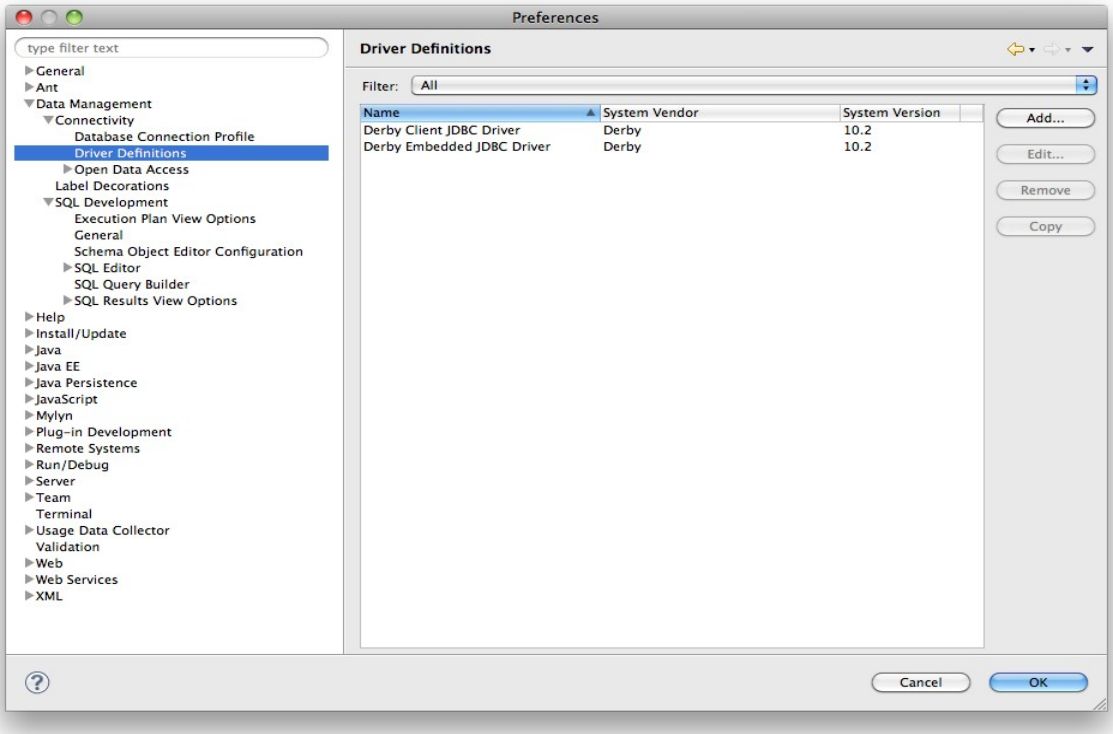
1.1. Télécharger le driver JDBC d'une base de données Derby:

URL: http://archive.apache.org/dist/db/derby/db-derby-10.2.1.6/db-derby-10.2.1.6-lib.zip
(ou db-derby-10.2.1.6-lib.tar.gz)

Dézipper db-derby-10.2.1.6-lib.zip dans « C:\ », le chemin obtenu du driver est:
« C:\db-derby-10.2.1.6-lib\lib\derby.jar ».

1.2. Configurer un driver Derby dans Eclipse:

Sélectionner: Eclipse → Préférences → Data Management → Connectivity → Driver Definitions.
Cliquer sur le bouton « Add » (à droite):

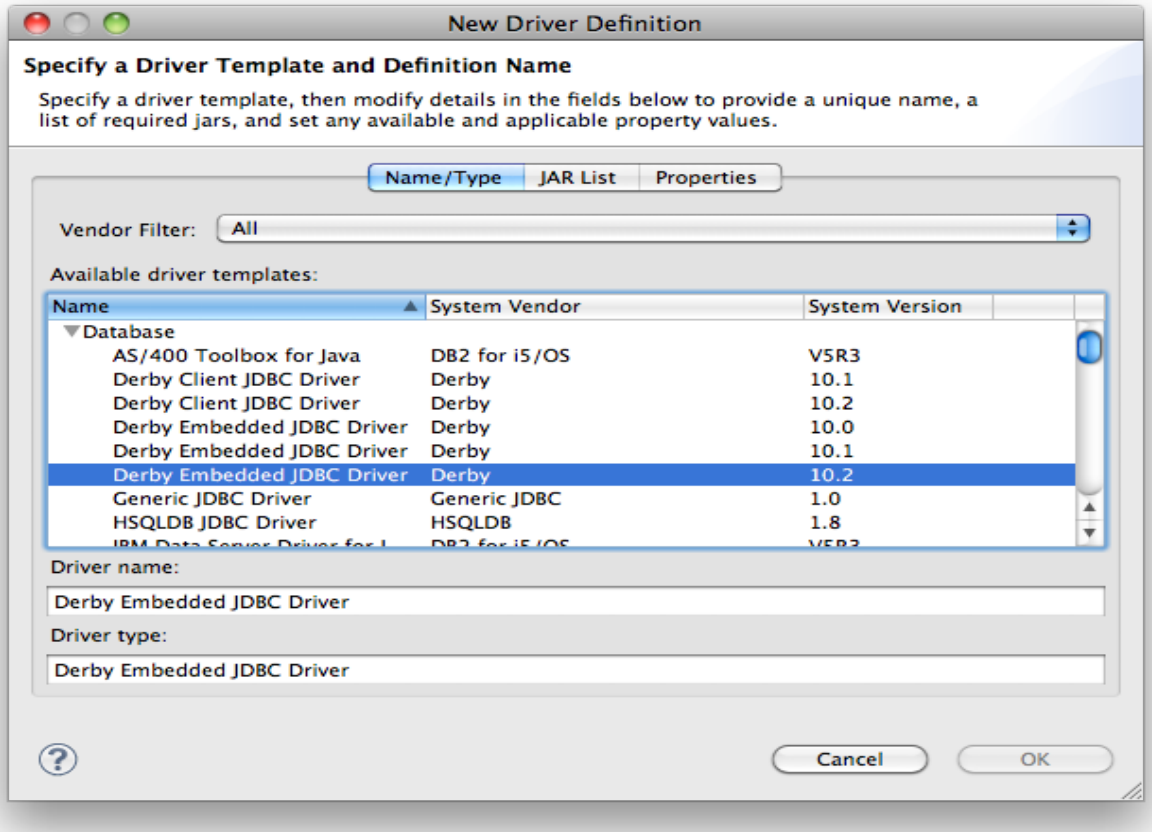


Une boite de dialogue « New Driver Definition » s'ouvre.

Cliquer sur l'onglet Name/Type et sélectionner "Derby Embeded JDBC Driver " version 10.2.

Note : Pourquoi « Embeded Derby » ?

Utiliser Embeded Derby permet la création de bases de données directement dans le système de fichiers (sous forme d'un dossier) sans passer par la configuration du serveur de base de données.



Ensuite, Cliquer sur l'onglet « JAR List » et sélectionner « derby.jar ».
Un clic sur le bouton Edit JAR/Zip ouvre l'explorateur des fichiers.
Sélectionner le chemin vers le fichier "derby.jar" : « C:\db-derby-10.2.1.6-lib\lib\derby.jar », OK.
Cliquer sur OK pour fermer la fenêtre « Preferences ».

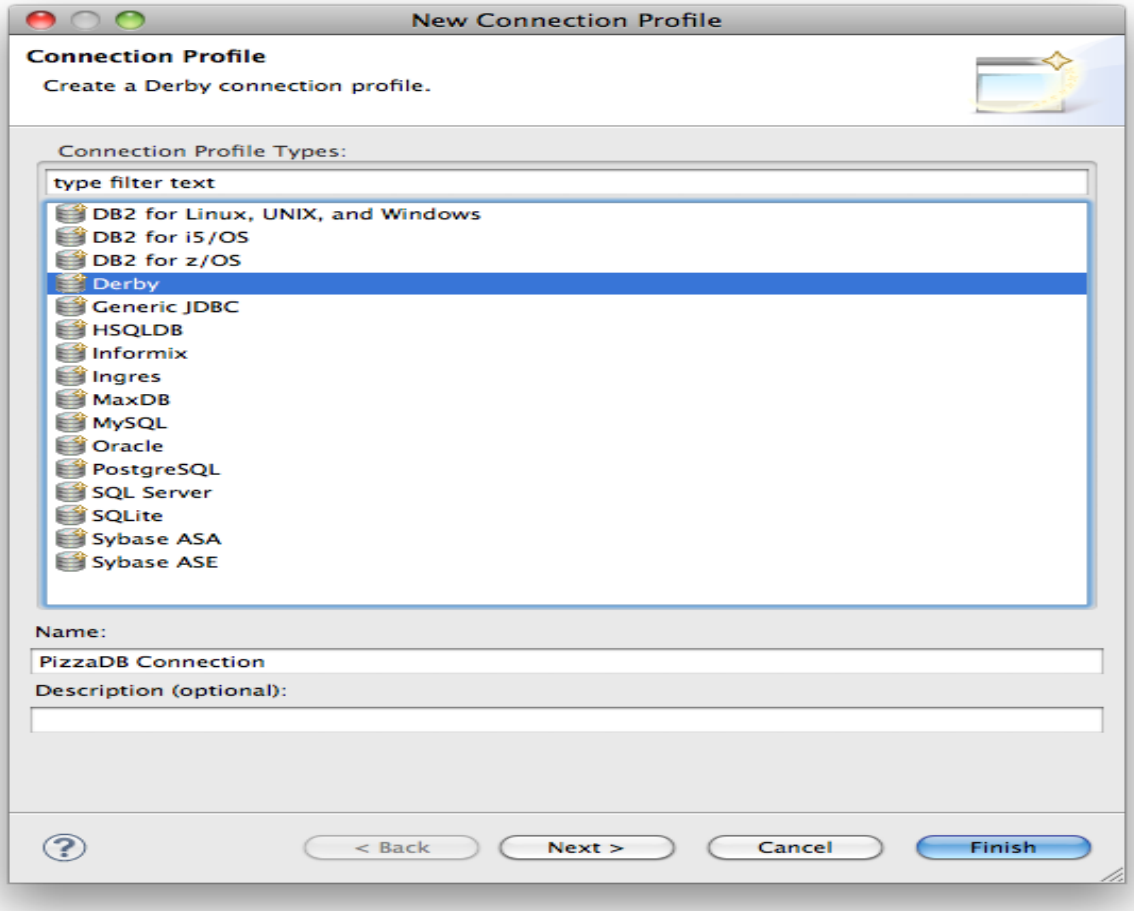
1.3. Création de la base de données PizzaDB:

L'onglet (view Eclipse) « Data Source Explorer », qui se trouve à coté de l'onglet « Servers », permet de gérer des bases de données, les drivers disponibles, les connecteurs possibles (un connecteur par base de donnée créée). Un tel connecteur permet de créer et ensuite d'accéder directement à la base de donnée, de voir les tables et leur contenu, d'exécuter des requêtes SQL (SQL Scrapbook).

Dans, la suite, on va utiliser l'onglet « Data Source Explorer » pour créer la base de données suivante:
- nom : PizzaDB
- user name : pizza
- password : pizza

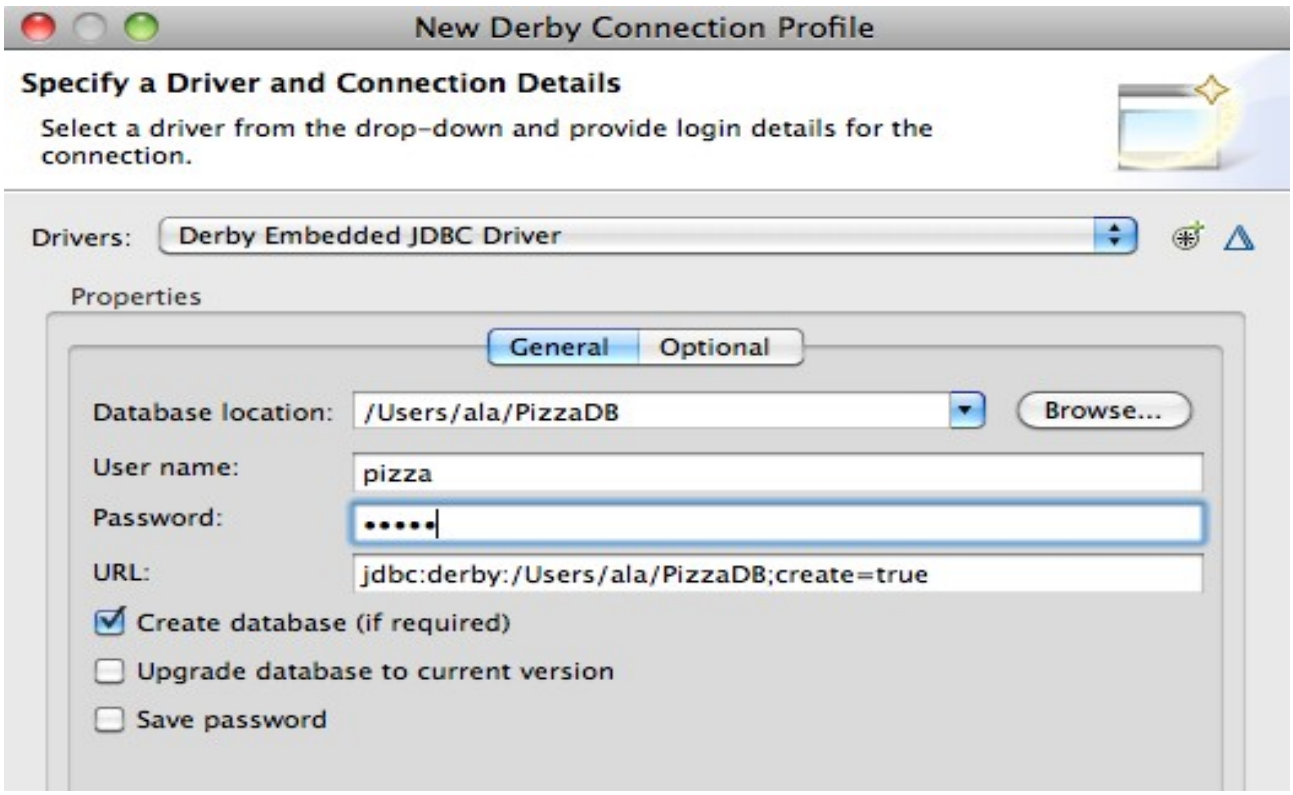
Pour créer une base de donnée, il faut cliquer droit sur « Database Connections » puis « New ».

Sélectionner "Derby", entrer le nom pour le profile de connexion (au choix): par exemple « PizzaDB Connexion », puis cliquer sur "Next":



Dans le champ « Drivers », sélectionner le driver Derby configuré à l'étape 1.2 : "**Derby Embedded JDBC Driver**".

Un clic sur l'onglet "General", puis sélectionner un dossier de destination pour la base de données: «C:\PizzaDB», le nom du répertoire «**PizzaDB**» est le nom de la base de données.



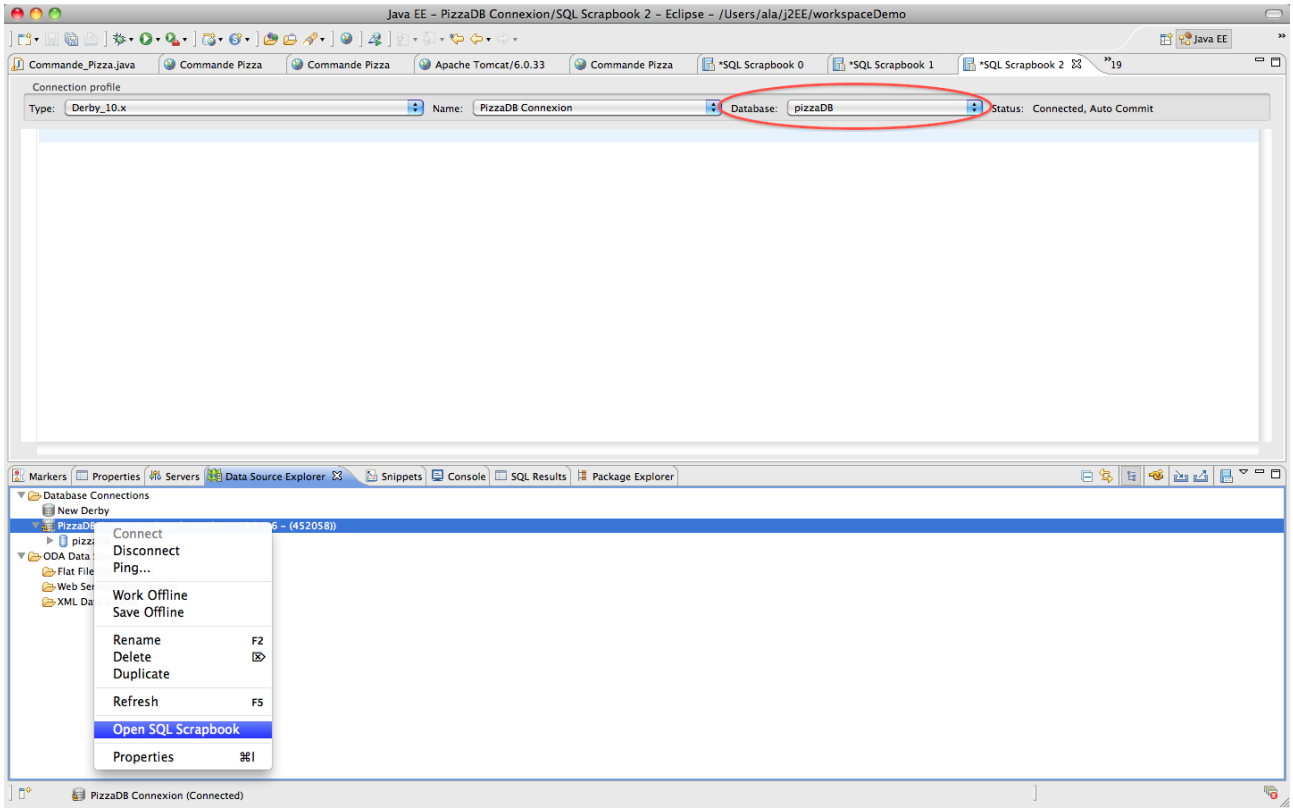
Enter un nom d'utilisateur dans "User name: **pizza**" et un mot de passe dans "Password: pizza" (entrées optionnelles).

Ne rien changer dans le champ "URL". Ce champ change automatiquement avec le nom de la base, c'est cette URL qui va identifier la base dans les classes java (PizzaFacade.java, ...).

Cocher la case "Create Database" et "Connect when the wizard completes"

1.4. Création des Tables de la base de données PizzaDB:

Dans l'onglet « Data Source Explorer », cliquer droit sur « PizzaDB Connexion » et sélectionner « SQL Scrapbook »: un éditeur SQL s'ouvre.

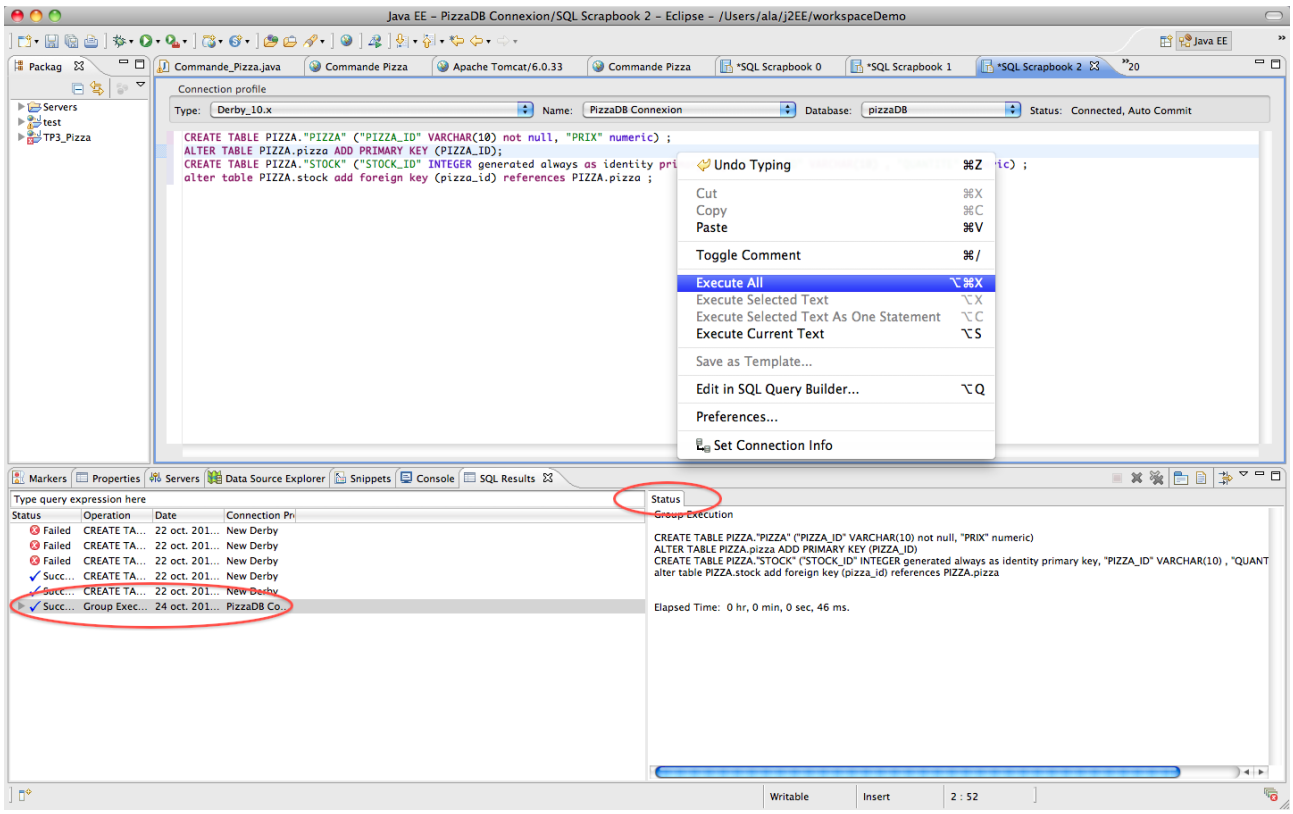


Pour exécuter des commandes SQL sur la base *pizzaDB*, sélectionner «Database: pizzaDB» dans la liste déroulante en haut à droite.

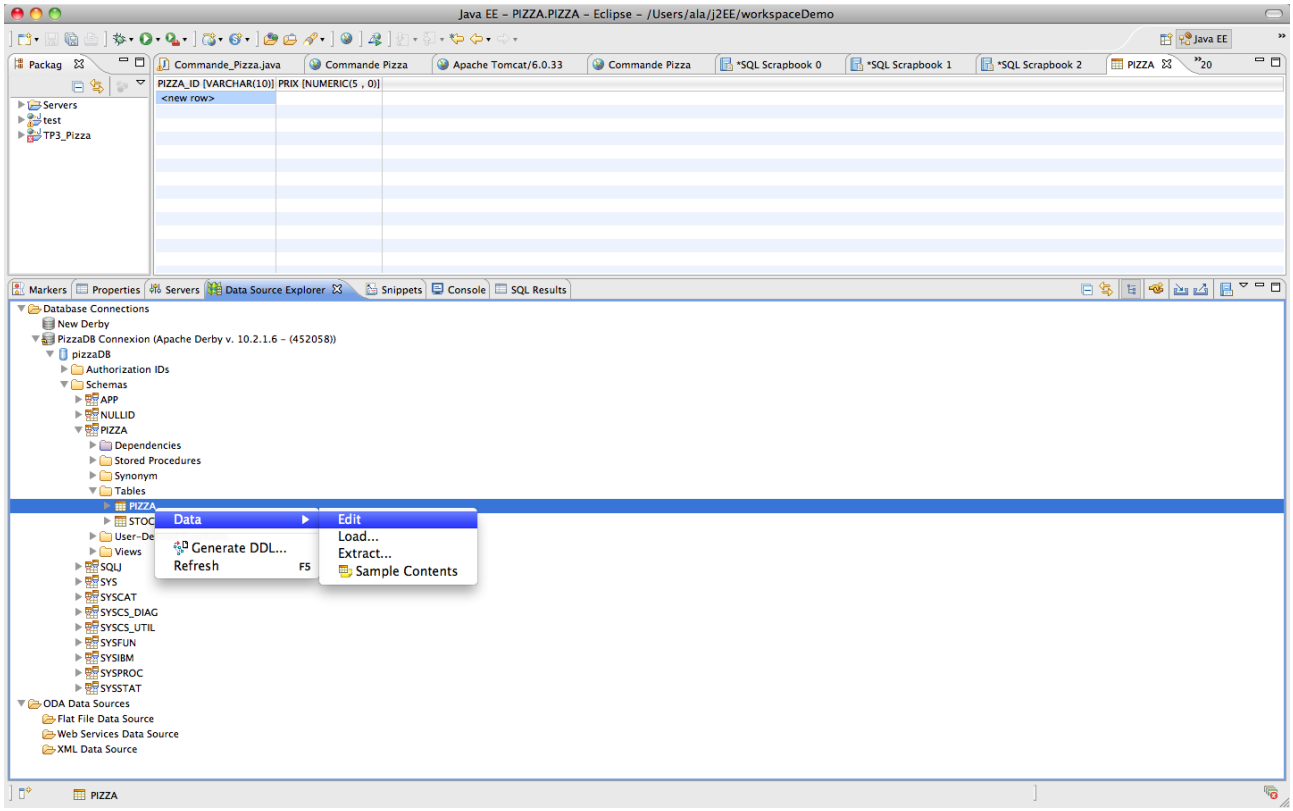
Entrer les commandes SQL de création des tables *PIZZA* et *STOCK*:

```
CREATE TABLE PIZZA."PIZZA" ("PIZZA_ID" VARCHAR(10) not null, "PRIX" numeric) ;
ALTER TABLE PIZZA.pizza ADD PRIMARY KEY (PIZZA_ID);
CREATE TABLE PIZZA."STOCK" ("STOCK_ID" INTEGER generated always as identity
primary key, "PIZZA_ID" VARCHAR(10) , "QUANTITE" numeric) ;
alter table PIZZA.stock add foreign key (pizza_id) references PIZZA.pizza ;
```

Pour exécuter ces commandes, cliquer droit à l'intérieur de l'éditeur et sélectionner «Execute All»:



Si Le code SQL est bien exécuté, l'onglet "SQL Result" affiche "success" et l'onglet "status" affiche le code exécuté.
Il est possible d'afficher le contenu des tables en descendant dans l'arborescence suivante:
Database Connections → PizzaDB Connexion → pizzaDB → Schemas → PIZZA → Tables.
Un clic doit permet d'afficher un menu «data» pour visualiser, éditer, … les données des tables:



2. L'API Java/JDBC et Mapping Objet-relationnel:

L'objectif de cette partie est de mettre en place un site web pour l'administration et la commande de pizzas :

- une pizza est déterminée par son type et son prix à l'unité.
- un stock (quantité) est donné pour chaque type de pizza
- une commande se définit par un type de pizza, une quantité, le total de la commande, l'email de confirmation de la commande.

2.1. Servlet AdminPizza:

- Créer un nouveau projet web dynamique: *TP3_Pizza-war*.

- Dans ce projet, on va mettre en place à l'aide de JDBC, un Mapping Objet-relationnel (Hibernate que vous avez déjà vu et EJB 3 sont basés sur cette technique de programmation).
Comme rappel, vous pouvez consulter le lien http://fr.wikipedia.org/wiki/Mapping_objet-relationnel. Un Mapping Objet-relationnel permet d'associer un Objet JavaBean à chaque ligne d'une table de la base de données. Par exemple, le contenu de chaque ligne de la table `PIZZA` sera récupéré (coté Java) dans un JavaBean de la classe `Pizza.java` (voir le code de la classe `Pizza.java` en annexe A et dans l'annexe B la méthode `findPizzaById()` de la classe `PizzaFacade.java`).
Ainsi, le contenu de toutes les lignes de la table `PIZZA` sera récupéré sous la forme d'une liste de JavaBeans de la classe `Pizza.java` (voir la méthode `getListPizzas()` de la classe `PizzaFacade.java` en annexe).

- Dans le répertoire «*src*», créer les classes JavaBean mappant les tables de la base *PizzaDB*: pour cela, il suffit de compléter la classe *Pizza.java* (getter et setter pour tous les champs) et de créer la classe *Stock.java* de la même manière.

- Dans le répertoire «*src*», créer les classes (*Façade*) utilisant JDBC pour accéder à la base de données. Ces classes nécessitent l'ajout du driver JDBC *«derby.jar»* dans le répertoire *WEB-INF/lib*.
La classe *PizzaFacade.java* est fournie en Annexe, elle implémente les méthodes suivantes:

1. la méthode *getConnection()* permet de récupérer une connexion vers la Base *PizzaDB,*
2. la méthode ***create(Pizza p)*** permet d'exécuter une requête SQL de type «Insert» afin de stocker une nouvelles ligne dans la Table *PIZZA*.
3. la méthode ***findPizzaById(String pizzaId)*** permet d'exécuter une requête SQL de type «Select» afin de récupérer la ligne de la Table *PIZZA* correspondant à une clef primaire égale à la valeur du paramètre «*pizzaId*». Le contenu de la ligne sélectionnée est retourné sous la forme d'un JavaBean *Pizza.java*.
4. la méthode ***findAllPizzas()*** permet d'exécuter une requête SQL de type «Select» afin de récupérer toutes les lignes de la Table *PIZZA (*sous la forme d'une liste de JavaBeans *Pizza.java).*

- En s'inspirant de la classe *PizzaFacade*, créer la classe *StockFacade* proposant des méthodes *findAllStocks()* et *create(Stock s) analogues au méthodes de PizzaFacade.*

- Copier la Servlet *AdminPizza.java* à partir du TP2. Ensuite, afin d'utiliser la base *PizzaDB* comme source de données, adapter le code de *AdminPizza.java* pour utiliser les Façades *PizzaFacade* et *StockFacade* développées dans les questions précédentes. (Rappel: dans le TP2, les Façades utilisaient la «*Session*» comme source de données, dans ce TP on utilise une base de données).

- Tester la Servlet *AdminPizza.java*: Attention, il faut déconnecter «PizzaDB Connexion » avant de démarrer le serveur Tomcat (dans l'onglet «Data Source Explorer», cliquer droit sur «PizzaDB Connexion» puis sélectionner «Disconnect».
Utiliser la Servlet pour créer des nouveaux types de pizza, puis vérifier que les tables (*Pizza* et *Stock*) de la base de données ont été mises à jour.

2.2. Gestion des commandes de pizzas:

2.2.1. Servlet *CommandePizza*:

En utilisant le code fourni en Annexe C, créer une Servlet «*CommandePizza.java»* permettant de gérer les commandes de pizzas. Le code fourni implémente uniquement les fonctionnalités suivantes*:*

- Afficher la liste des stocks de pizzas.
- Saisir des nouvelles commandes de pizzas.

Dans la suite du TP (section 2.2.2), on va ajouter ces fonctionnalités:

- Mettre à jours le stock après chaque commande
- Refuser les commandes pour un stock insuffisant et refuser les commandes pour un type de pizza inexistant.

Ajouter la table *COMMANDE* à la base de données *PizzaDB*: une commande se définit par un id *(COMMANDEID)* généré automatiquement, un type de pizza *(PIZZAID)*, une quantité, le total de la commande, l'email de confirmation (ne pas oublier la contrainte sur la clé étrangère `PIZZAID`).

```
CREATE TABLE PIZZA."COMMANDE"(
"COMMANDEID"INTEGER generated always as identity primary key,
"PIZZAID" VARCHAR(10),
"QUANTITE" numeric,
"TOTAL" numeric,
"EMAIL"varchar(20)
);
alter table PIZZA.commande add foreign key(PIZZAID) references PIZZA.PIZZAID;
```

Dans le répertoire «*src*», créer la classe JavaBean correspondant à la table *COMMANDE,* puis ajouter la Façade *CommandeFacade.java* contenant l'implémentation des méthodes appelées par la Servlet *CommandePizza.java.* Enfin, utiliser la Servlet pour créer des commandes et vérifier le résultat.

2.2.2. Développements:

Compléter le code de la Servlet ***CommandePizza et des Façades*** pour implémenter les fonctionnalités suivantes :

1. Refuser les commandes pour un type de pizza inexistant et refuser les commandes pour un stock insuffisant. Pour cela, ajouter le code ci-dessous à la Servlet *CommandePizza* et ajouter à *StockFacade* une méthode `findStockByPizzaId(String pizzaId)` qui retourne un JavaBean *Stock* contenant la ligne de la table *Stock* correspondant au paramètre `pizzaId`.

```
Stock stock = stockFacade.findStockByPizzaId(type);
if (stock == null) {
  out.println("Nous ne prenons pas les commandes pour les pizzas "+type+"<br/>");
} else if (stock.getQuantite() < quantite) {
  out.println("Commande non effectuée. Vous demandez "+quantite+" pizzas et le stock est de
    " + stock.getQuantite() + "<br/>");
} else {
 ...
}
```

2. Mettre à jour le stock après chaque commande:
   Pour cela, la Servlet *CommandePizza* doit mettre à jour le (JavaBean) *stock* puis appeler la méthode suivante de la classe *StockFacade*:

```
	public void update(Stock stock) {
		Connection connection = getConnection();
		String query = "update stock set quantite = " + stock.getQuantite()
				+ " where pizza_id = '" +  stock.getPizzaId() + "'";

		try {
			connection.createStatement().executeUpdate(query);
			connection.close();
		} catch (SQLException e) {
			e.printStackTrace();
		}
	}
```

Annexe:

A. Code des JavaBeans*:*

- Code du JavaBean *Pizza.java:*

```
package entityPizza;
public class Pizza implements Serializable {
    private String pizzaId;
    private Integer prix;

    public Pizza() {}

    //TODO generate getters and setters for all fields
    ...
    public void setPizzaId(String pizzaId) {
        this.pizzaId = pizzaId;
    }
    public void setPrix(Integer prix) {
        this.prix = prix;
    }
}
```

- Code du JavaBean *Stock.java:*

```
package entityPizza;
public class Stock implements Serializable {
    private static final long serialVersionUID = 1L;

    private Integer stockId;
    private Integer quantite;
    private String pizzaId;

    public Stock() {
    }
   //TODO generate getters and setters for all fields
    ...
}
```

- Code du JavaBean *Commande.java:*

```
package entityPizza;
public class Commande implements Serializable {
    private Integer commandeId;
    private Integer quantite;
    private Integer total;
    private String email;
    private String pizzaId;
    public Commande() {
    }
   //TODO generate getters and setters for all fields
    ...
}
```

B. Code de la Façade *PizzaFacade.java:*

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;import java.util.List;
public class PizzaFacade {

	public void create(Pizza p) {
		Connection connection = getConnection();
		String query = "insert into pizza(pizza_id,prix) ";
		query = query + "values ('" + p.getPizzaId() + "'," + p.getPrix() + ")";

		try {
			connection.createStatement().executeUpdate(query);
			connection.close();
		} catch (SQLException e) {e.printStackTrace();}

	}

	public Pizza findPizzaById(String pizzaId) {
		Pizza pizzaBean = null;
		Connection connection = getConnection();
		String query = "SELECT * FROM Pizza WHERE pizza_id = '" + pizzaId + "'";
		try {
			ResultSet rs = connection.createStatement().executeQuery(query);
			if (rs.next()) {
				pizzaBean = new Pizza();
				// / Get the data from the row using the column name
				pizzaBean.setPizzaId(rs.getString("PIZZA_ID"));
				pizzaBean.setPrix(rs.getInt("PRIX"));
			}
			 connection.close();
		} catch (SQLException e) {e.printStackTrace();}

		return pizzaBean;

	}

	public List<Pizza> findAllPizzas() {
		ArrayList<Pizza> pizzas = new ArrayList<Pizza>();
		try {
			Connection connection = getConnection();
			String query = "SELECT * FROM Pizza";
			ResultSet rs = connection.createStatement().executeQuery(query);

			// Fetch each row from the result set
			while (rs.next()) {
				Pizza pizzaBean = new Pizza();
				// Get the data from the row using the column name
				pizzaBean.setPizzaId(rs.getString("PIZZA_ID"));
				pizzaBean.setPrix(rs.getInt("PRIX"));
				pizzas.add(pizzaBean);
			}
			connection.close();
		} catch (SQLException e) {
			e.printStackTrace();
			 return null;
		}

		return pizzas;
	}

	private Connection getConnection() {
		try {
			Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
			//TODO modifier la ligne suivante avec le chemin de votre base de données
			return DriverManager.getConnection("jdbc:derby:[à remplacer par l'URL votre base]/PizzaDB"
			,"pizza", "pizza");
		} catch (Exception e) {
			e.printStackTrace();
			return null;
		}

	}
}
```

C. Code de la Servlet *CommandePizza.java (*gestion des commandes de pizzas*)*:

```
package gestionClient;
import entityPizza.CommandeFacade;
import entityPizza.Pizza;
import entityPizza.PizzaFacade;
import entityPizza.Stock;
import entityPizza.StockFacade;
public class CommandePizza extends HttpServlet {
          private StockFacade stockFacade = new StockFacade();
          private CommandeFacade commandeFacade = new CommandeFacade();
          private PizzaFacade pizzaFacade = new PizzaFacade();

       protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Commande Pizza</title>");
            out.println("</head>");
            out.println("<body>");
            List lStocks = stockFacade.getListStocks();
            for (Iterator it = lStocks.iterator(); it.hasNext();) {
                 Stock stock = ((Stock) it.next());
                out.println("Type : <b>" + stock.getPizzaId() + " </b> ");
                out.println("Stock : " + stock.getQuantite() + "<br/>");
            }
            out.println("<h1>Choisissez votre pizza : </h1>");
            String type=request.getParameter("type");
            if (type!=null) {
                try {
                        Pizza pizza = pizzaFacade.find(type);
                        int quantite=new Integer(request.getParameter("quantite"));
                        //On ajoute une commande dont le prix total est égale à:
                        int total = quantite * pizza.getPrix();
                        Commande commande = new Commande();
                        commande.setTotal(total);
                        commande.setPizzaId(request.getParameter("type"));
                        commande.setQuantite(new Integer(request.getParameter("quantite")));
                        commande.setEmail(request.getParameter("email"));
                        //On persiste la commande en base de données
                        commandeFacade.create(commande);
                        //Commande effectuée, donc on recharge la page
                        response.sendRedirect("Commande_Pizza");
                } catch (Exception ex) {
                    out.println("Commande non effectuée");
                    ex.printStackTrace();
                }

            }
                out.println("<form method='POST' action='Commande_Pizza'>");
                out.println("Type: <input type='text' name='type'><br/>");
                out.println("Quantité: <input type='text' name='quantite'><br/>");
                out.println("Email: <input type='text' name='email'><br/>");
                out.println("<input type='submit'><br/>");
                out.println("</form>");
                out.println("<a  href='Admin_Pizza'> Administration pizza</a>");
                out.println("</body>");
                out.println("</html>");
           } finally {
            out.close();
           }
       }
       protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
            processRequest(request, response);
        }
       protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
            processRequest(request, response);
        }
}
```