# Models
# Exercise

**Chapter:** **Models. EJB**

**Theme:** **Using Web Dynpro to access EJBs**

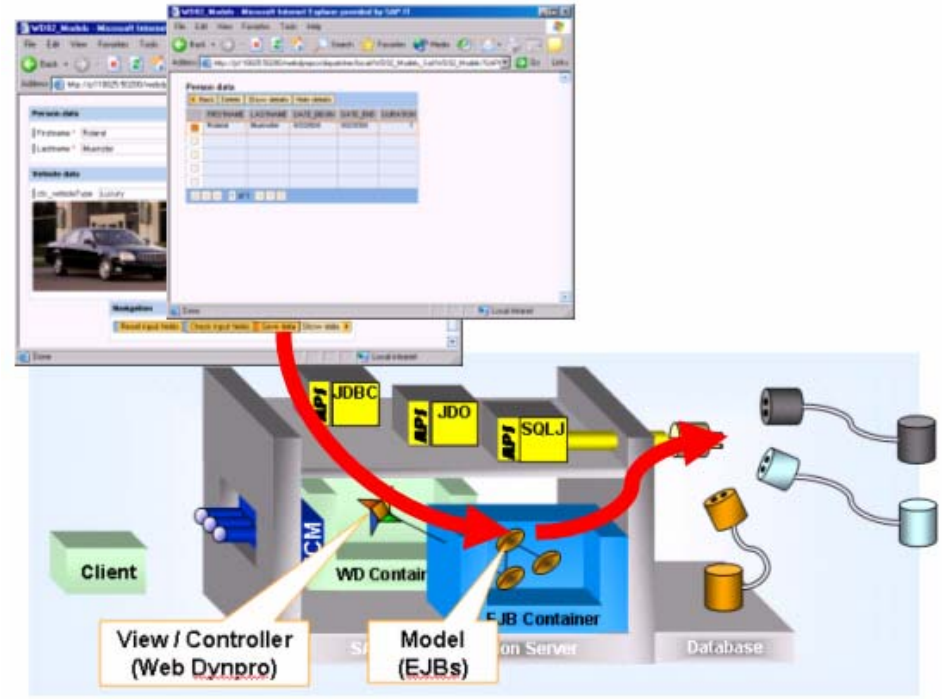
At the end of this Exercise, you are able to:

- Access EJB functionality from Web Dynpro.

## 1 Development Objectives

*Accessing EJBs*
The following exercise shows, how to design, implement, deploy, and run a basic Web Dynpro application that accesses persistent data from an EJB application.

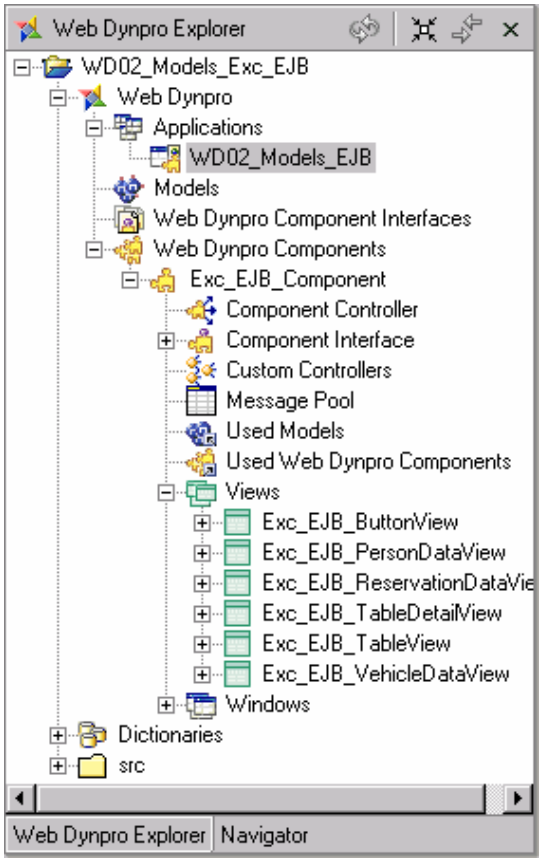
## 2 Result




As a result of this exercise, you complete a simple, structured Web application, which will add/display car reservations to/from a relational database You should be able to enter some reservation data. When pressing the SAVE button, the data is stored on the DB using a predefined stateless session bean. You should also be able to select the content of the database table and remove entries from the database.

## 3 Prerequisites

You have launched the SAP NetWeaver Developer Studio.
You have selected the Web Dynpro perspective.
You have opened the project *WD02_Models_Exc_EJB.*




For your convenience, you can start developing with a predefined Web Dynpro application.

The graphic on the left shows the predefined project structure of this exercise.

All contexts, views, data types, messages and so on are predefined.

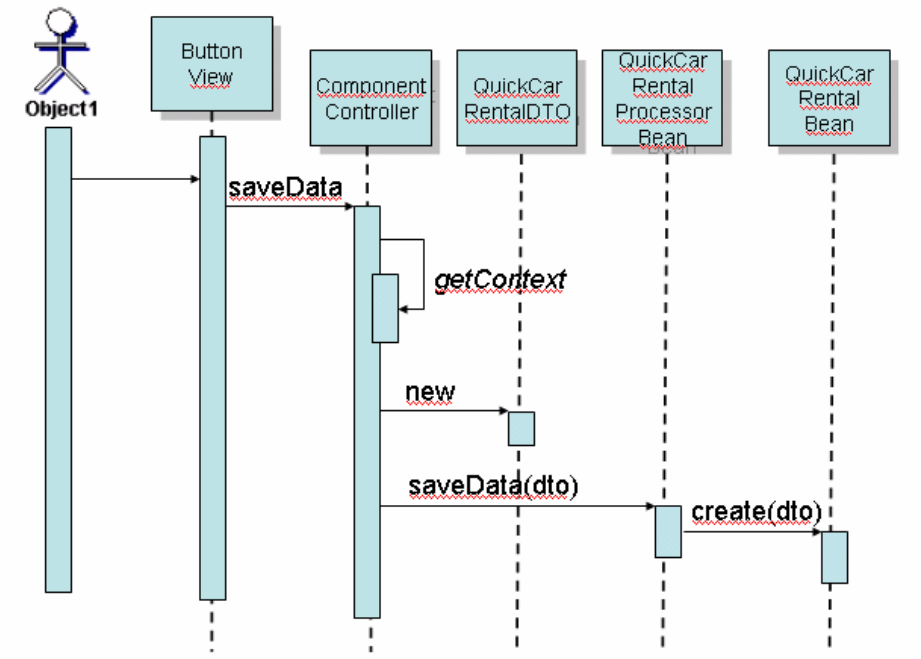
You can deploy and run the predefined WD application *WD02_Models_EJB.*

Your task is to connect the WD application to an existing EJB application:

- New data should be stored on the database via predefined EJB components
- Selected data should also be removed from the database

## 4 Overview: Developing

4-1 Deploy the predefined Dictionary.archive.
4-2 Deploy the predefined EJB components.
4-3 Customize the project settings.
  4-3-1 Add the EJB-jar file to the WD project.
  4-3-2 Define the Sharing Reference of your Web Dynpro Project. The syntax is:
  *<vendor name>/<name of the ear file without extension>*.
4-4 Complete the SAVE action
  4-4-1 Edit the implementation of the View Controller *Exc_EJB_ButtonView*. Uncomment the source code of the method *onActionSaveData(…)*.
  4-4-2 Edit the implementation of the Component Controller.
  Uncomment the import statements.
  Uncomment the source code of the methods *save_data(…)* and *initializeReservationBean(…)*.
  Uncomment the declaration of the variable *reservationBean*.




The graphics above illustrates, how the save action works. When the user chooses the Save button in the *ButtonView*, the *saveData()* method of the Component Controller is invoked. Since the context of the views (which contain all input field data) is mapped to the context of Component Controller, all input field values are available in the Component Controller context. After having received these context values, the Component Controller creates a Data Transfer Object (DTO). This DTO is passed to a Stateless Session Bean by calling the beans *saveData()* method. The Stateless Session Bean itself creates a Container Managed Entity Bean (CMP) using the DTO as parameter. The Web Application Server is responsible for storing the data on the database.

**Note: The code is totally predefined. You only have to navigate to the corresponding sections and uncomment the source code lines.**

## 5 Overview: Building, Deploying, and Running

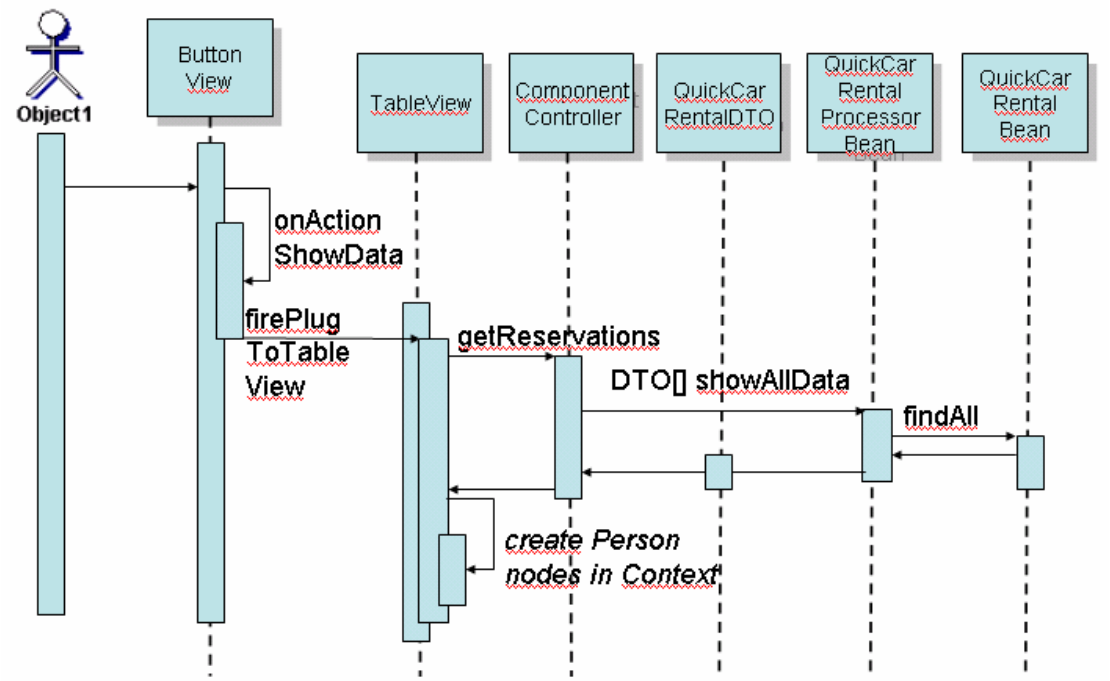
5-1 Deploy and run the Web Dynpro application
5-2 Open the SQLStudio to check the success.

## 6 Optional: Complete the SHOW and DELETE actions

6-1 Complete the SHOW action

6-1-1 Edit the implementation of the View Controller *Exc_EJB_TableView*. Uncomment the source code of the method *onPlugFromNavigationView(…)*.

6-1-2 Edit the implementation of the Component Controller. Uncomment the source code of the method *getReservations(…)*.



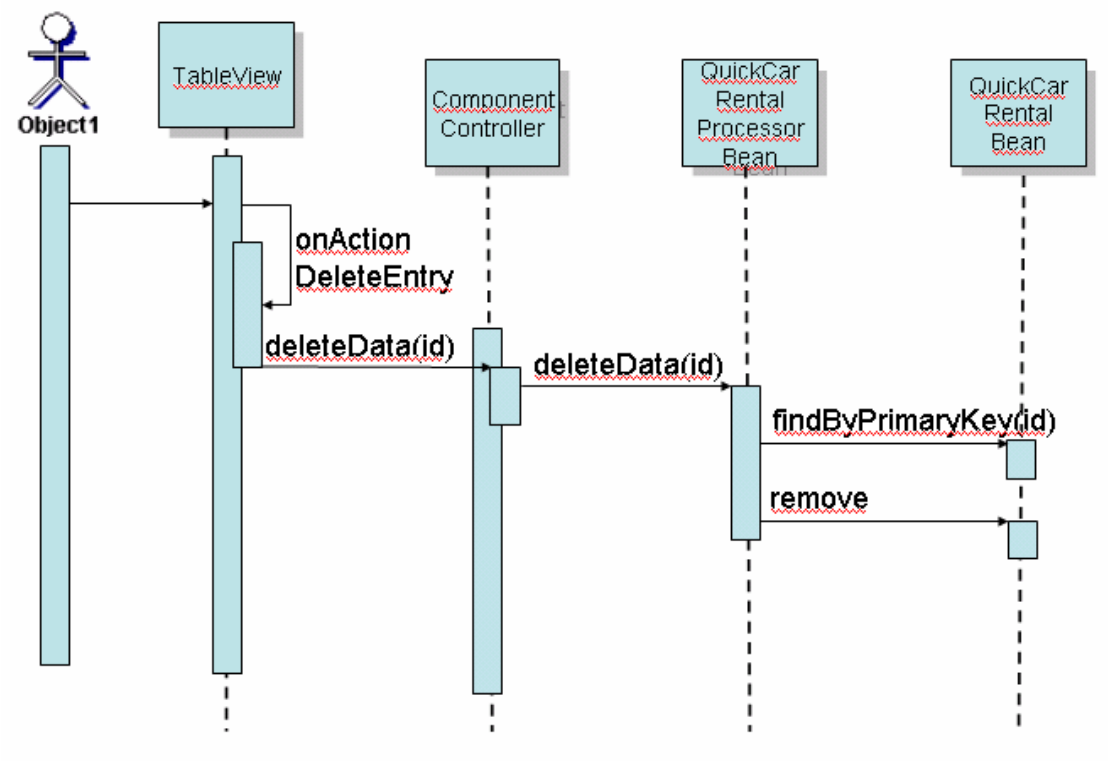

The graphics above illustrates, how the *show* action works:
When the user chooses the *Show*-button in the *ButtonView*, the *onActionShowData()* method of the *ButtonView* is invoked.
Within this method the Outbound Plug *firePlugToTableView* is fired, which navigates to the corresponding Inbound Plug *onPlugFromNavigationView*. Within this method, the *getReservations* method from the Component Controller is called. This method requests all available data from the database by calling the *showData* method of the *Stateless Session Bean* (This method calls an ejb-finder method of the corresponding CMP Entity Bean).
All available data is responded to the TableView using DTOs. For each DTO the TableView creates a new *Person* node in its context, which is then displayed in the table UI element automatically.
**Note: The code is totally predefined. You only have to navigate to the corresponding sections and uncomment the source code lines.**

6-2 Deploy and run the application.

6-3 Complete the DELETE action

6-3-1 Edit the implementation of the View Controller *Exc_EJB_TableView*. Uncomment the source code of the method *onActionDeleteEntry()*.

6-3-2 Edit the implementation of the Component Controller. Uncomment the source code of the method *deleteData(…)*.




The graphics above illustrates how the *delete* action works:
When the user chooses the *Delete* button in the *TableView*, the *onActionDeleteData()* method of the is invoked.
Within this method the method *deleteData(id)* from the Component Controller is called. This method calls the *deleteData(id)* method of the *Stateless Session Bean* This method then calls an ejb-findByPrimaryKey method of the corresponding CMP Entity Bean and removes the EJB instance in a second step. The Web Application Server is responsible to remove the corresponding database table entry.
**Note: The code is totally predefined. You only have to navigate to the corresponding sections and uncomment the source code lines.**

6-4 Deploy and run the application.