ADF – Делаем свое Псевдо Дерево (Step By Step)

Здравствуйте, уважаемые читатели!

Реализуя одну задачу и вдоволь намучившись со стандартным деревом и так и не добившись от него удовлетворительного результата, я решил пойти альтернативным путем…

Я решил сделать очень детальное описание, так сказать «Hard Hello World», и надеюсь, что данная статья будет полезна как начинающим, так и тем, кто уже владеет технологией ADF (для них, думаю, здесь будет много бесполезной информации).

Итак, приступим к реализации проекта…

Данный проект изначально писался в версии JDeveloper 12c, для версии 11g, смотрите примечания.

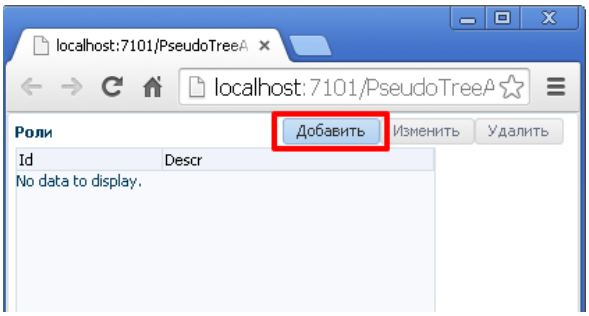
Демонстрационный проект для версии JDeveloper 11.1.1.7 можно скачать по следующей ссылке:
http://buhgalter-online.kz/files/j2ee/adf/PseudoTree.rar

Оглавление

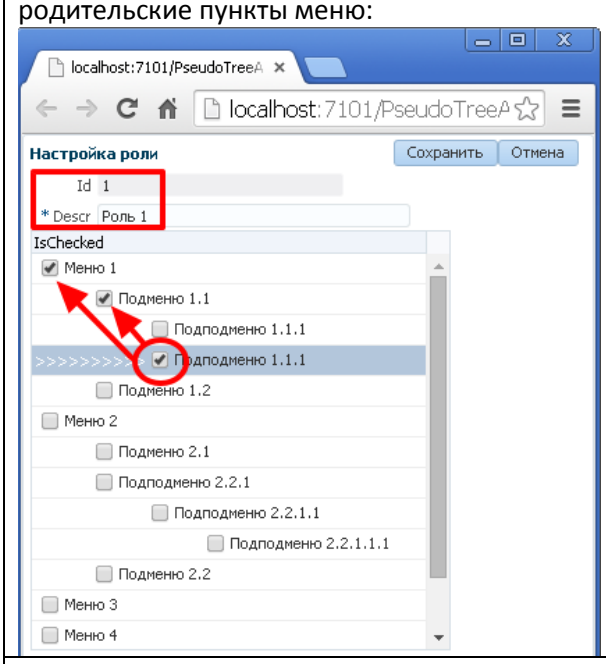
ADF – Делаем свое Псевдо Дерево (Step By Step) 1
Какой функционал я получил в итоге 2
Создаем схему и ее объекты 3
Сделаем полезную настройку в JDeveloper 3
Создание проекта 4
Создаем бизнес компоненты 5
Прикручиваем сиквенс 6
Создаем View Object по SQL-запросу 7
Создаем Task Flow и страницы 10
Настроим страницу roleList 12
Настроим страницу roleEdit 15
Наконец-то мы добрались до создания обещанного псевдо дерева 16
Настало время покодировать 22
Обходим ошибку InvalidOwnerException JBO-25030 24
Тестируем проект 25

Какой функционал я получил в итоге

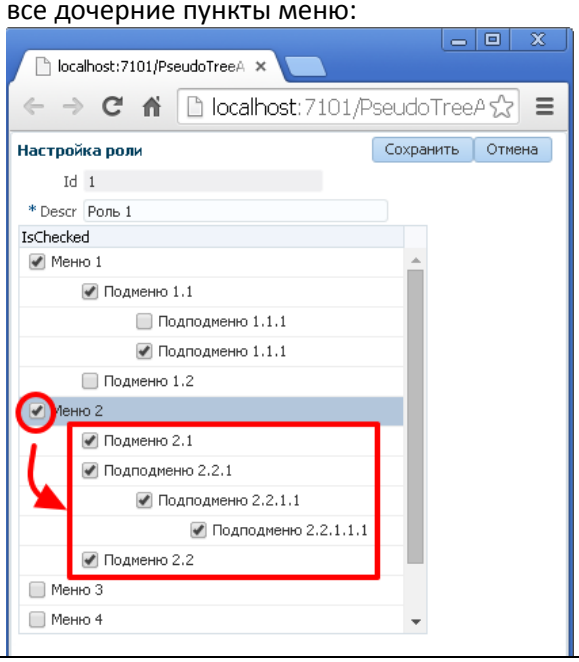
Добавление записи:



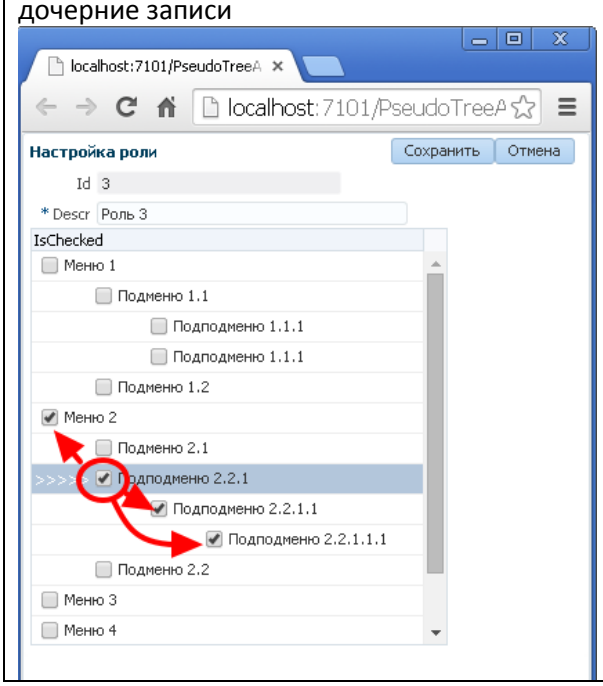
При отметке дочерней записи, нужно отметить все родительские пункты меню:



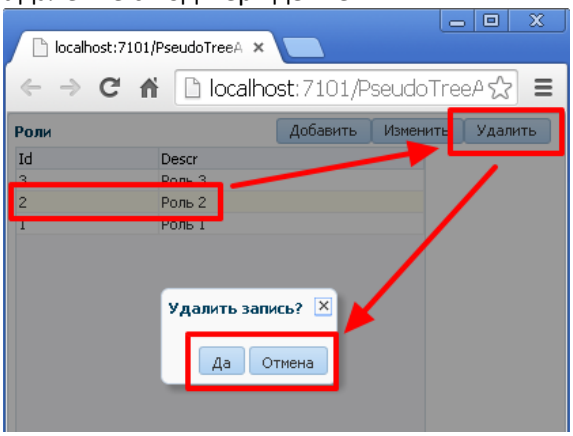
При отметке родительской записи, нужно отметить все дочерние пункты меню:



А в таком случае нужно отметить и родительские и дочерние записи

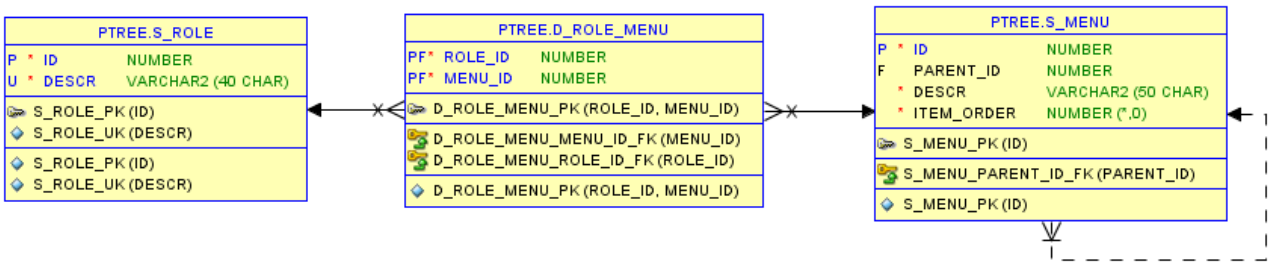


Удаление с подтверждением



Создаем схему и ее объекты

Создадим простую схему, состоящую из 3-х таблиц и одного сиквенса для таблицы S_ROLE:



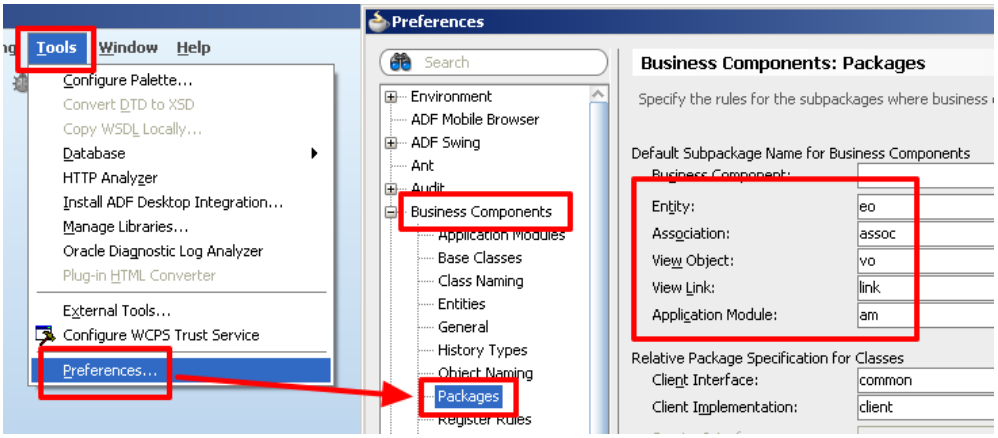
Где:

- S_ROLE – справочник ролей;
- S_MENU – меню представленное в виде дерева;
- D_ROLE_MENU – указывает на элементы меню доступные каждой роли.

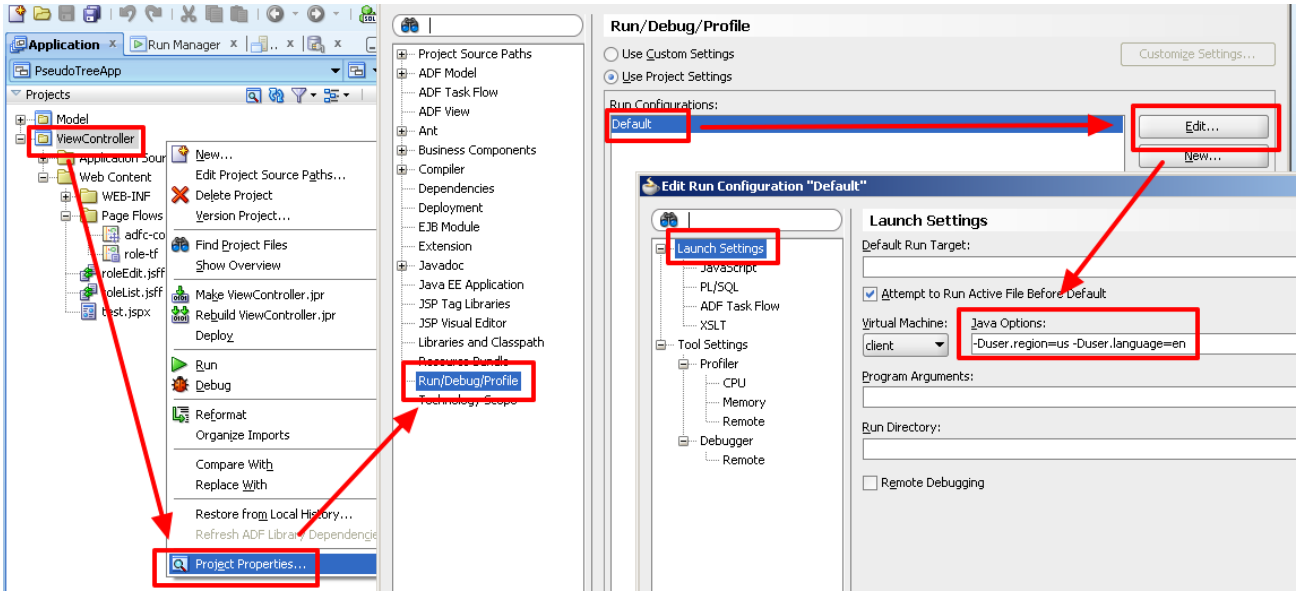
SQL-скрипт для создания схемы находится в файле «**CreateDB.sql**».

Сделаем полезную настройку в JDeveloper

Определим наименования пакетов для бизнес компонентов:



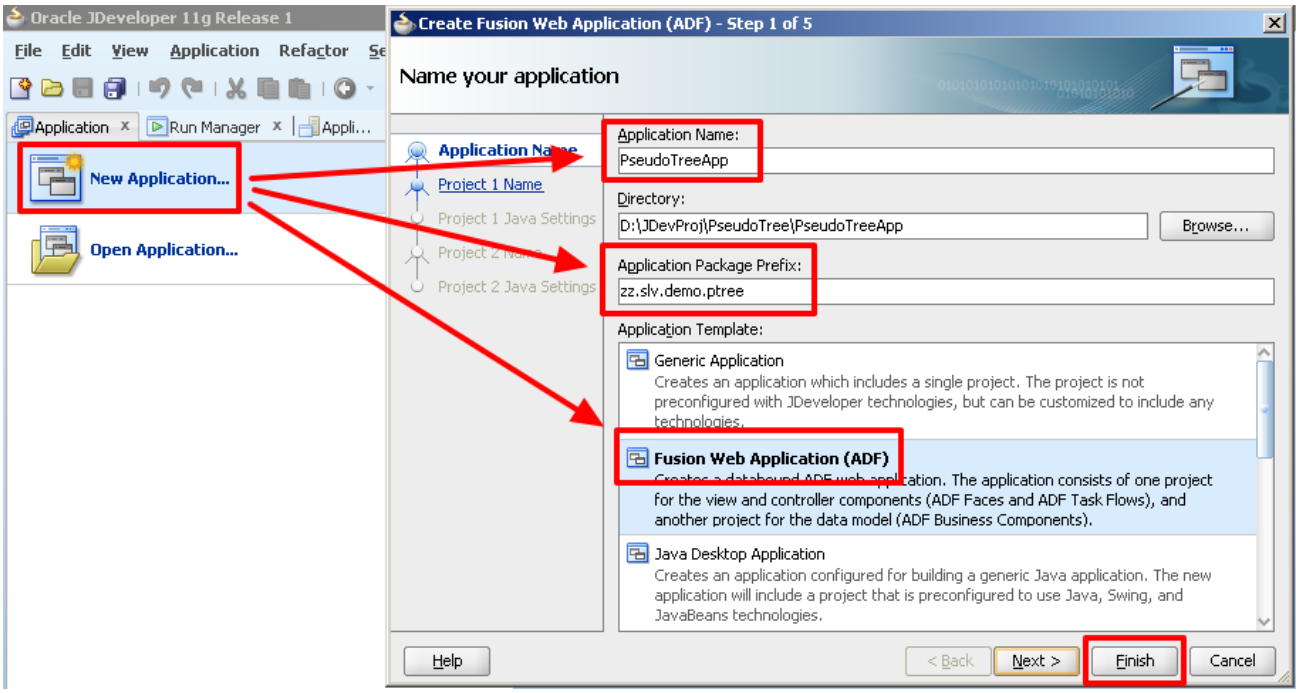
Возможно так же вам при NLS-ошибках помогут следующие опции:



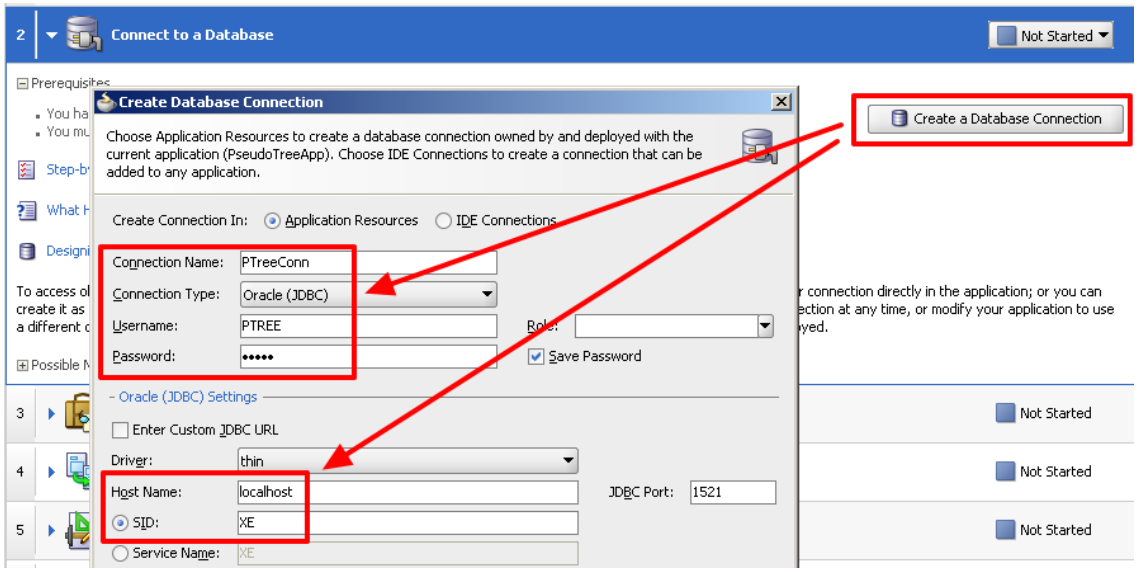
В некоторых случаях помогает установка (здесь же) параметра **-Duser.timezone="+06:00"**

Создание проекта

Создадим новое ADF-приложение (Fusion Web Application ADF):

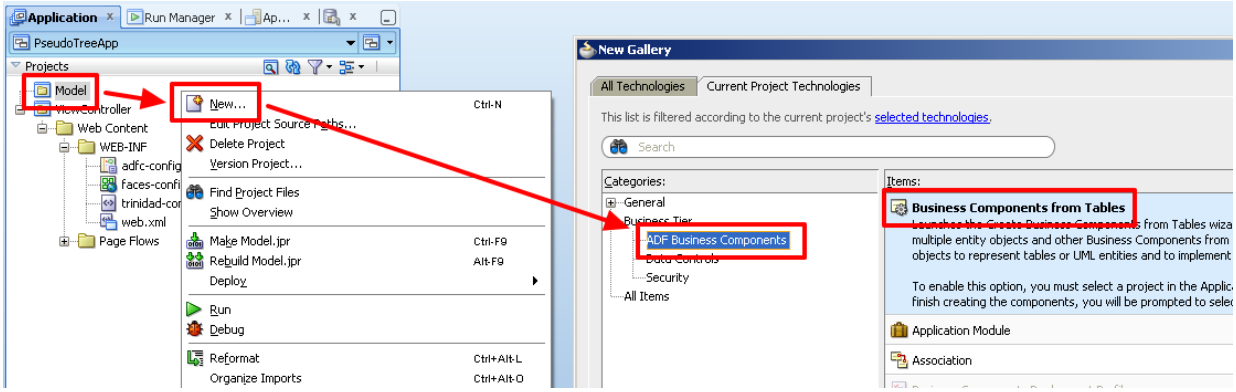


Создадим соединение с нашей БД:

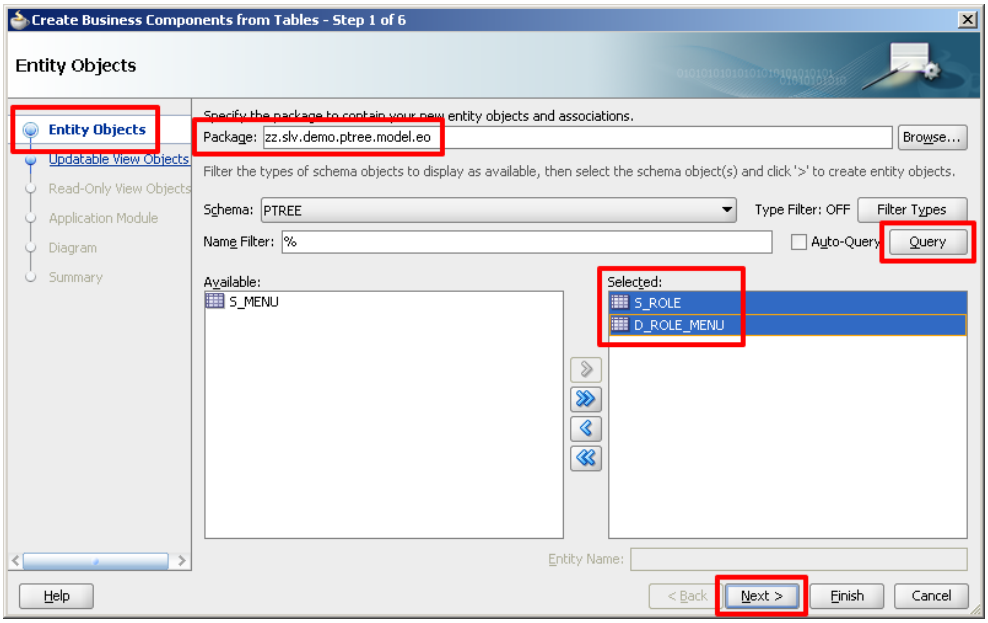


Создаем бизнес компоненты

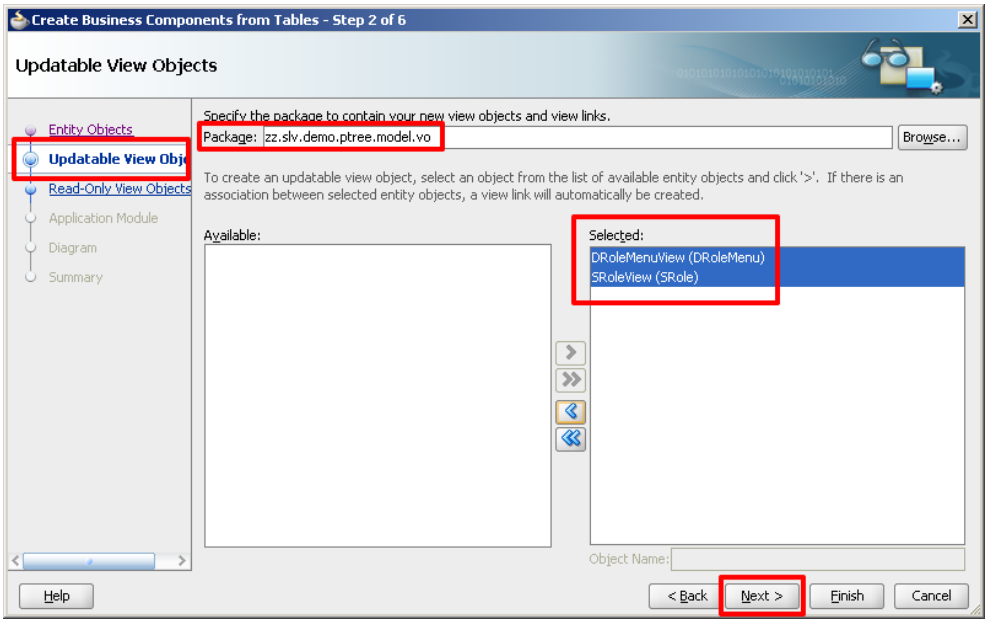
Для этой цели воспользуемся мастером:



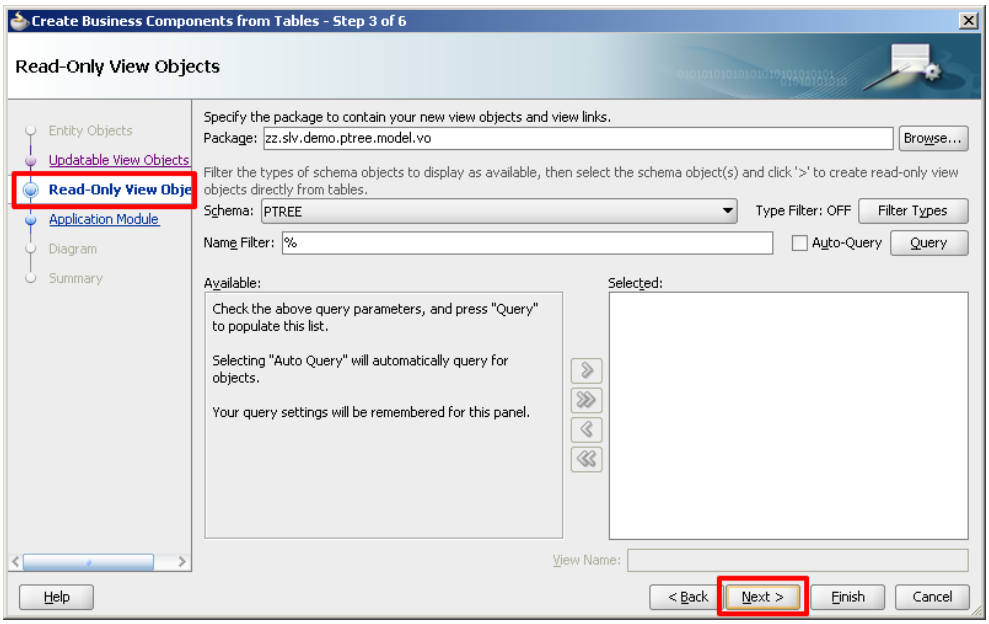
Выбираем таблицы, на которые будут созданы Entity Objects (выберем только **S_ROLE** и **D_ROLE_MENU**):



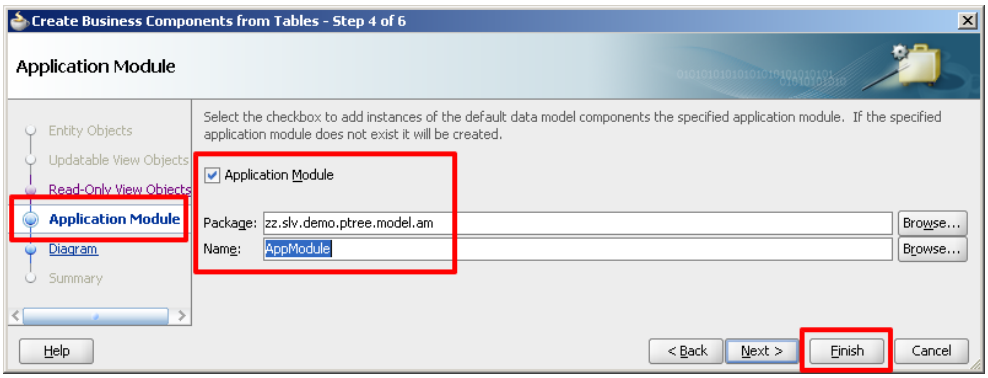
Создадим для этих 2-х объектов View Objects:



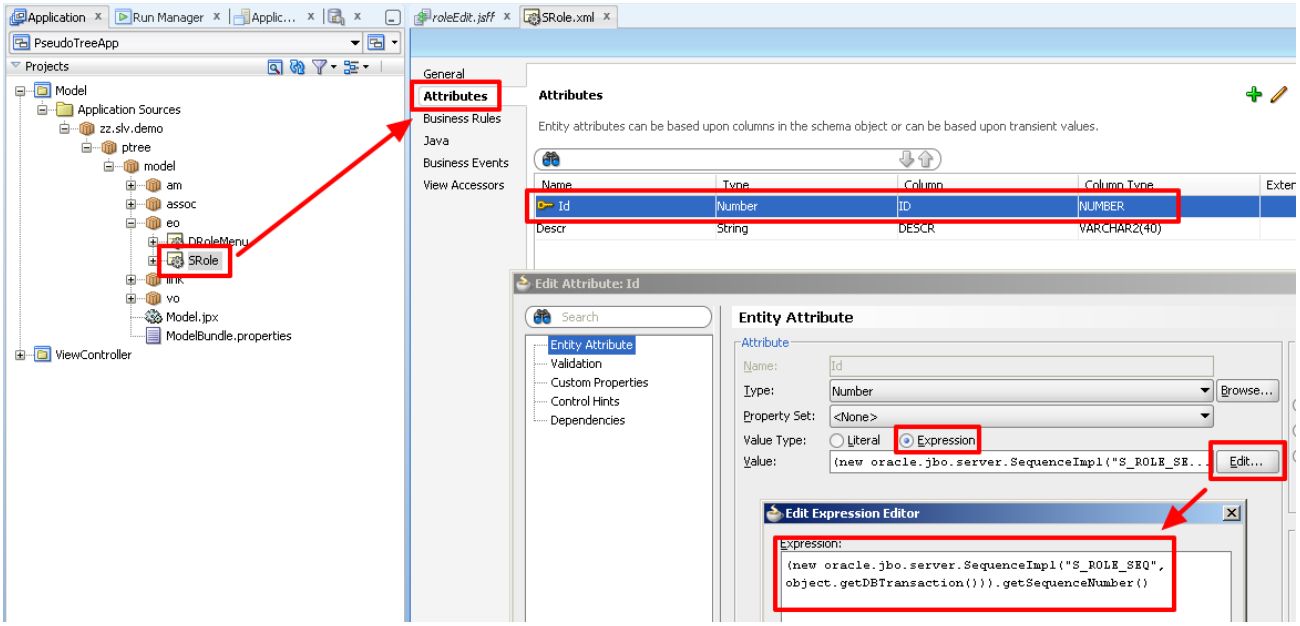
Следующий шаг просто пропустим:



И создадим Application Module (здесь можно сразу нажать Finish):

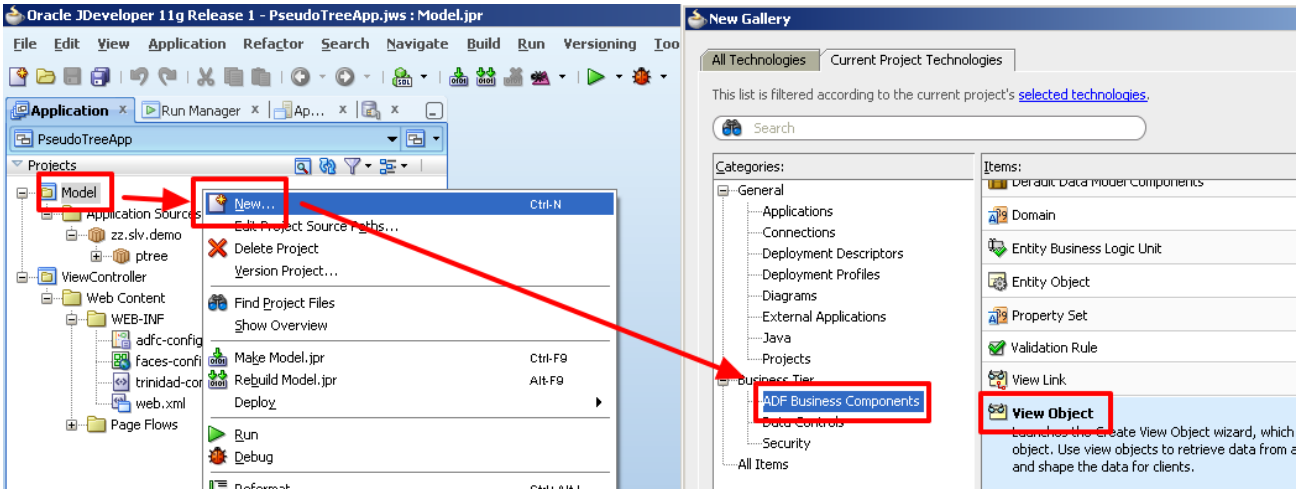


Прикручиваем сиквенс

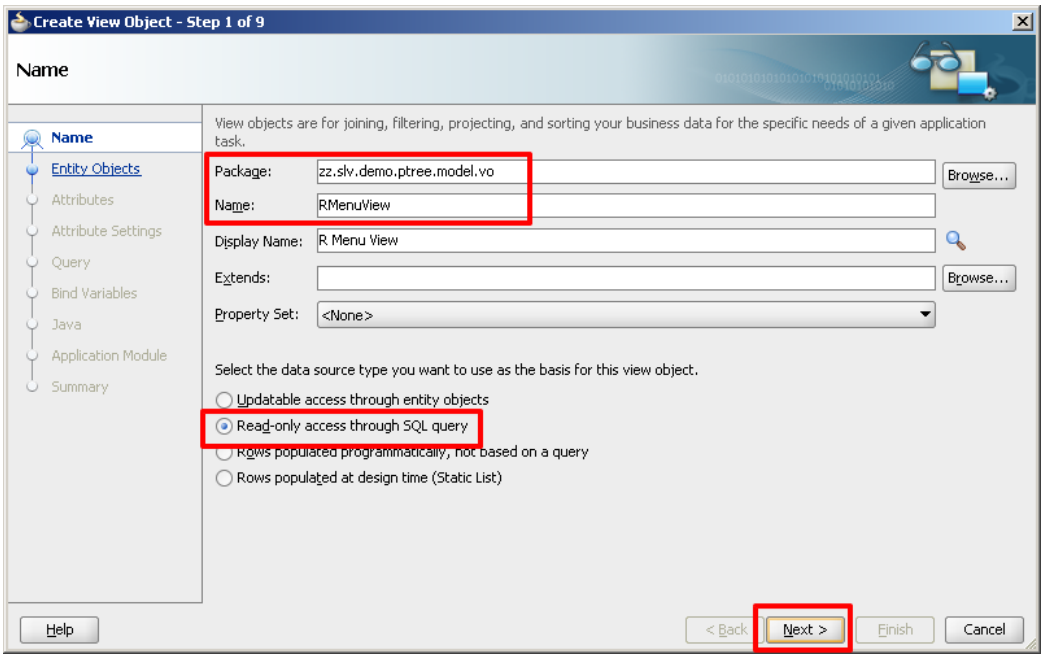


Создаем View Object по SQL-запросу

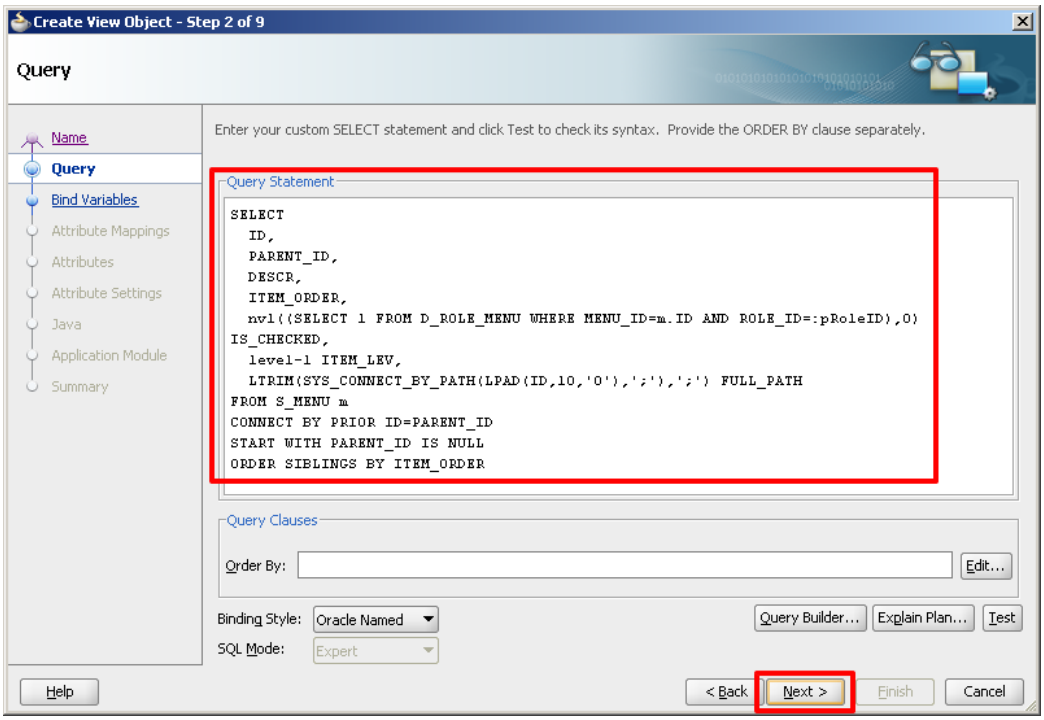
Создадим View Object, который будет отвечать за показ выбранных элементов меню для выбранной роли:



Говорим, что View Object будет строиться по SQL-запросу:

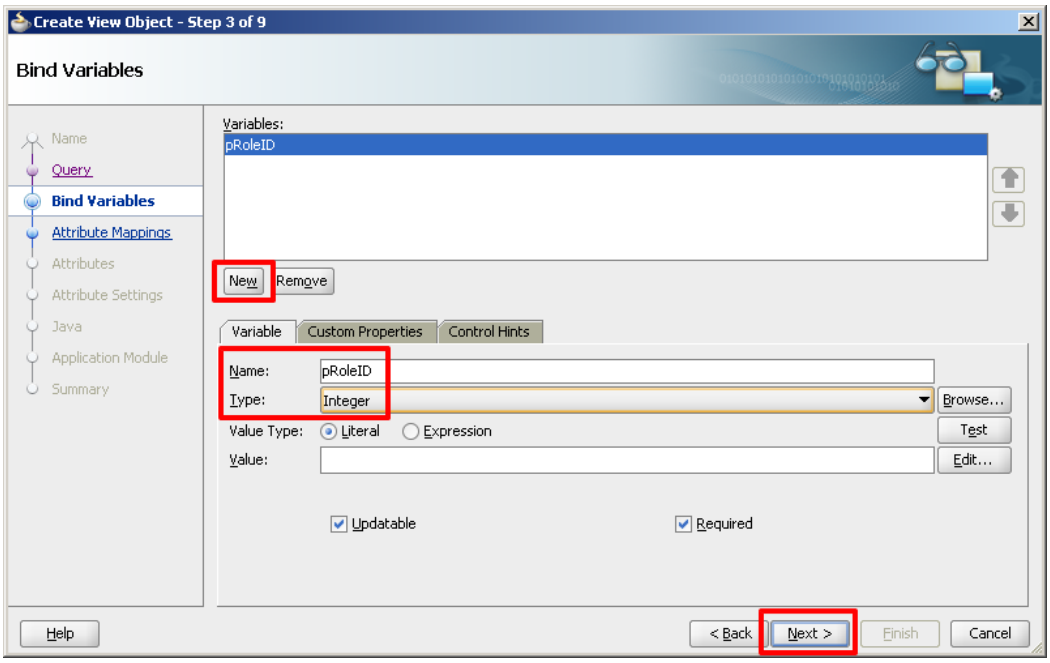


Напишем запрос при помощи старого, доброго **CONNECT BY PRIOR**:

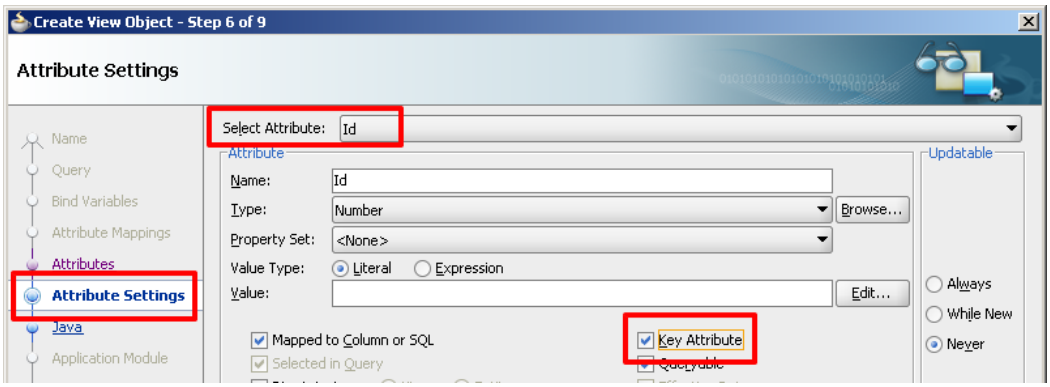


Данный запрос будет возвращать все элементы меню (все записи таблицы **S_MENU**), и вычисляемое поле **IS_CHECKED**, которое будет указывать на выбранные элементы для роли указанной при помощи параметра **pRoleID**. Здесь так же возвращаются 2 вспомогательных поля **ITEM_LEV** (показывает уровень вложенности) и **FULL_PATH** (хранящий полный путь до элемента, который содержит все идентификаторы родителей).

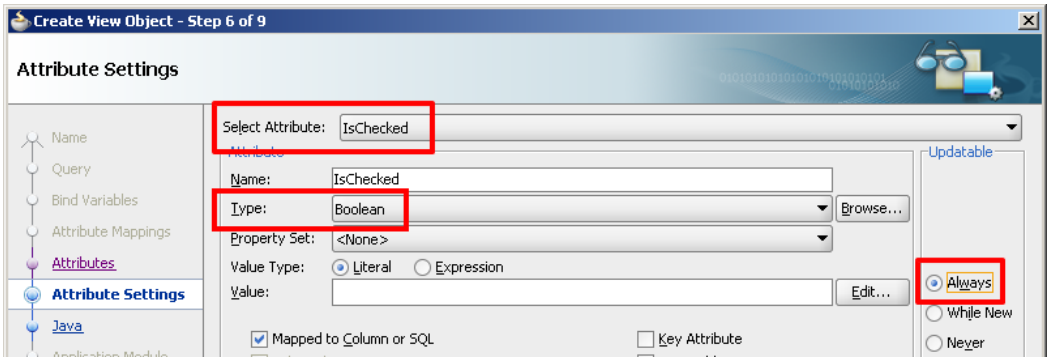
Определяем параметр, через который будем передавать ID роли:



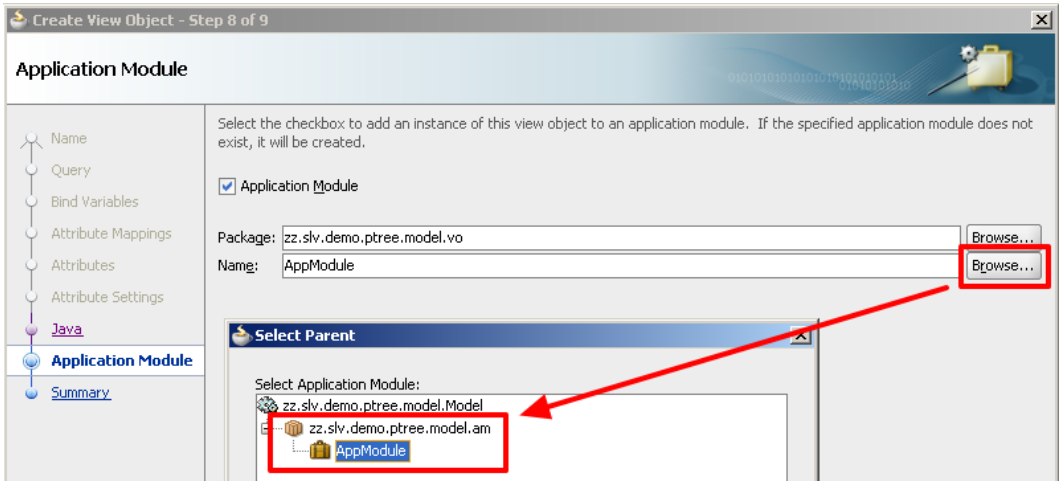
Ставим флажок **Key Attribute** для атрибута **ID**:



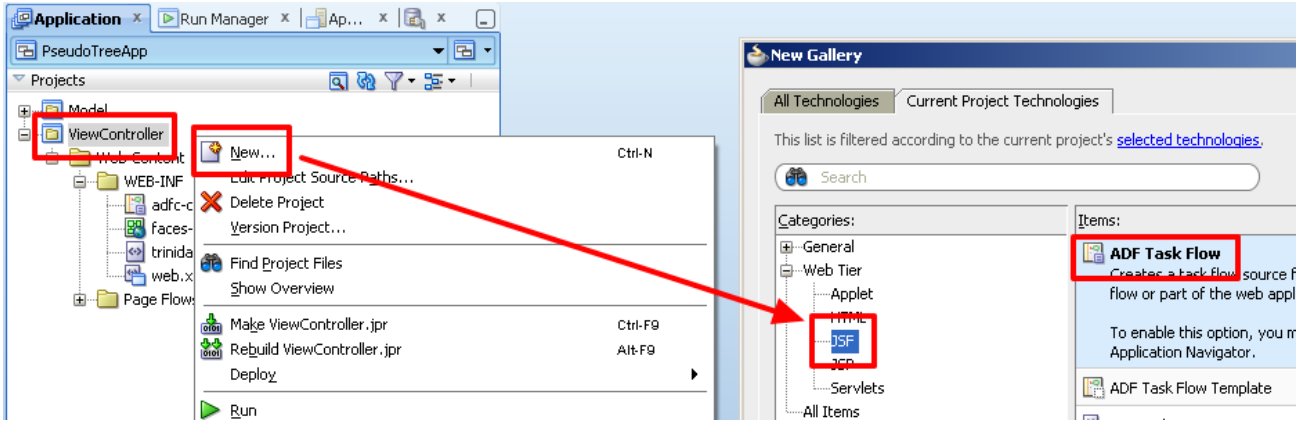
Для атрибута **IsChecked** ставим тип **Boolean** и **Updatable = Always**:

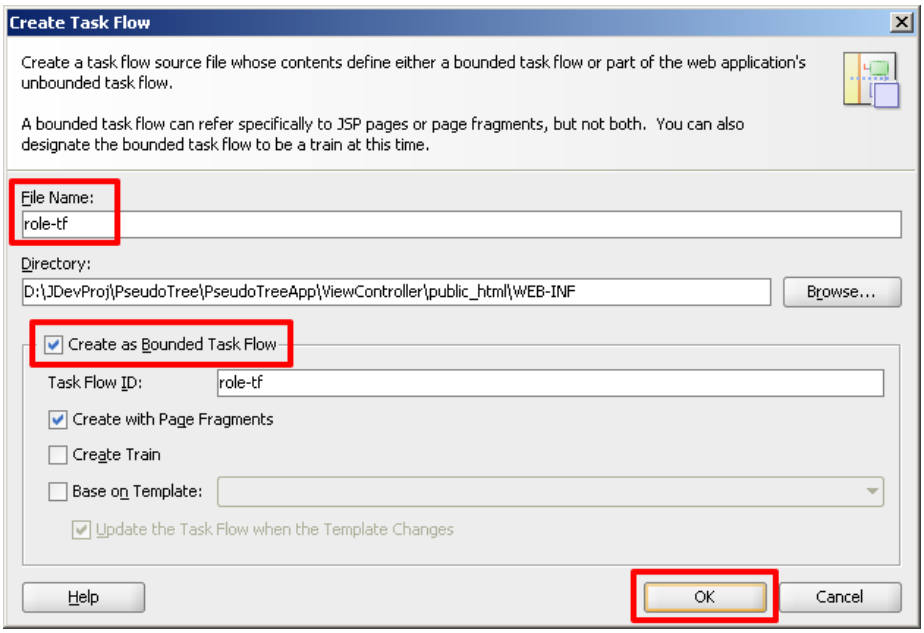


Сразу же добавляем новый View Object в наш Application Module (после можно нажать на Finish):

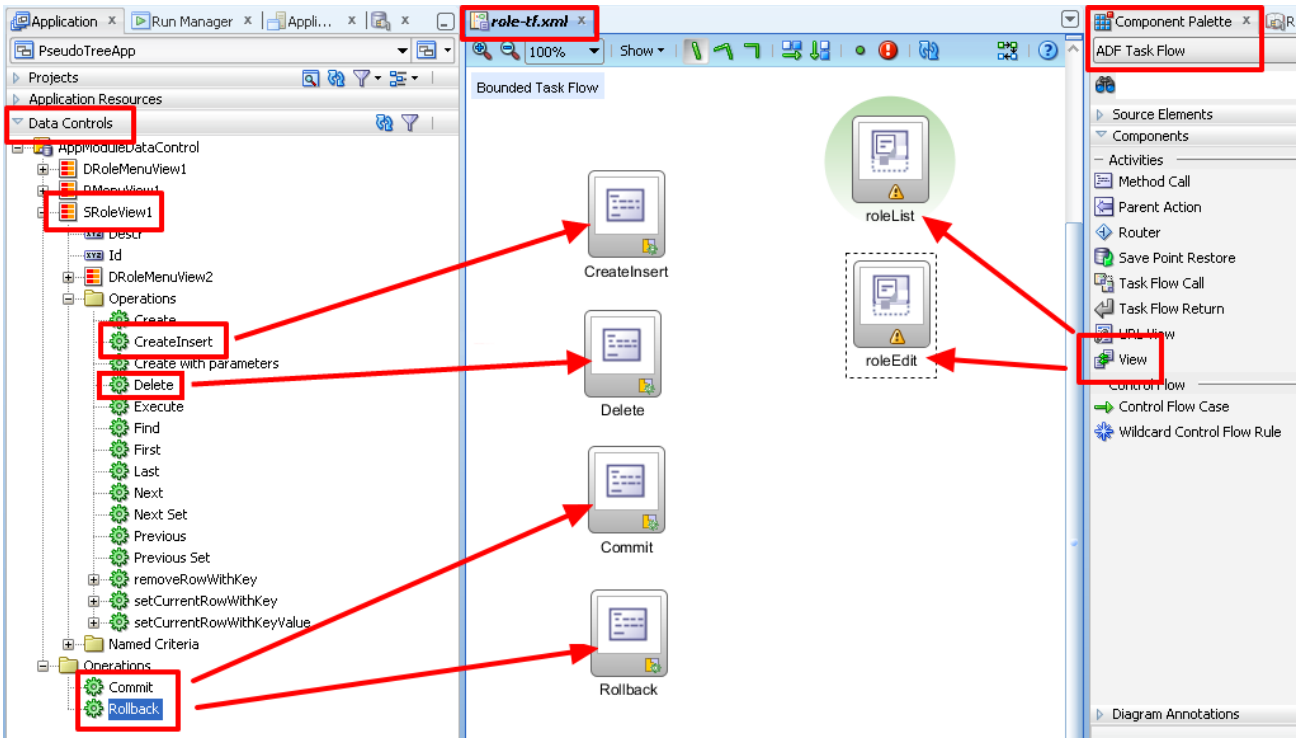


Создаем Task Flow и страницы

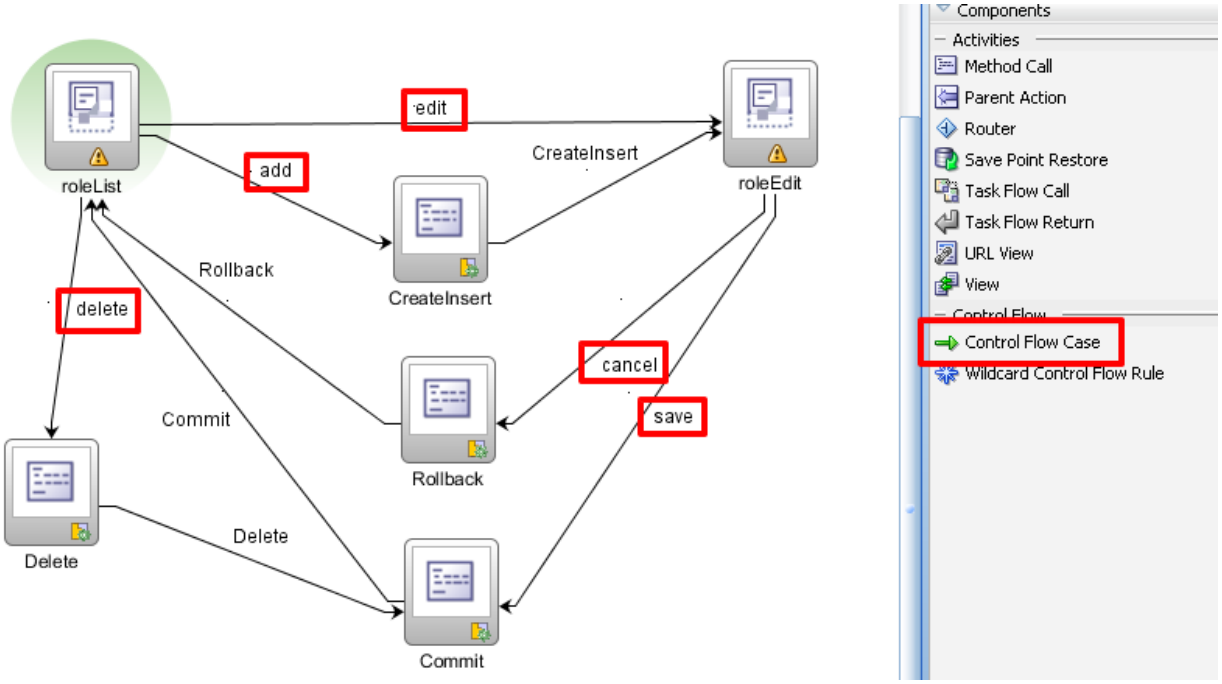




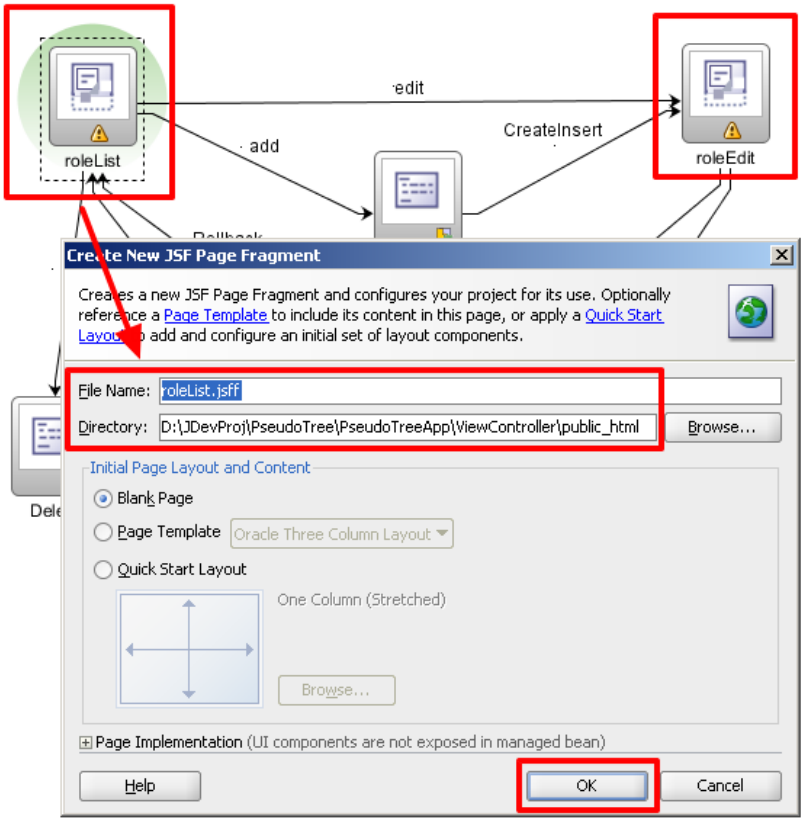
Переносим следующие элементы в Task Flow:



Настраиваем связи между элементами:

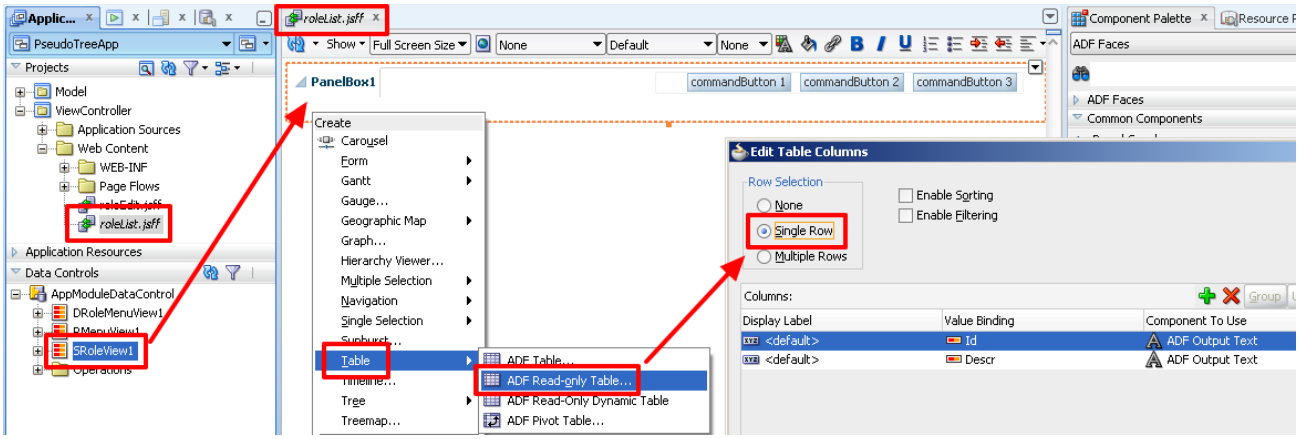


Двойными щелчками по **roleList** и **roleEdit** поочередно создаем JSFF-страницы (фрагменты):

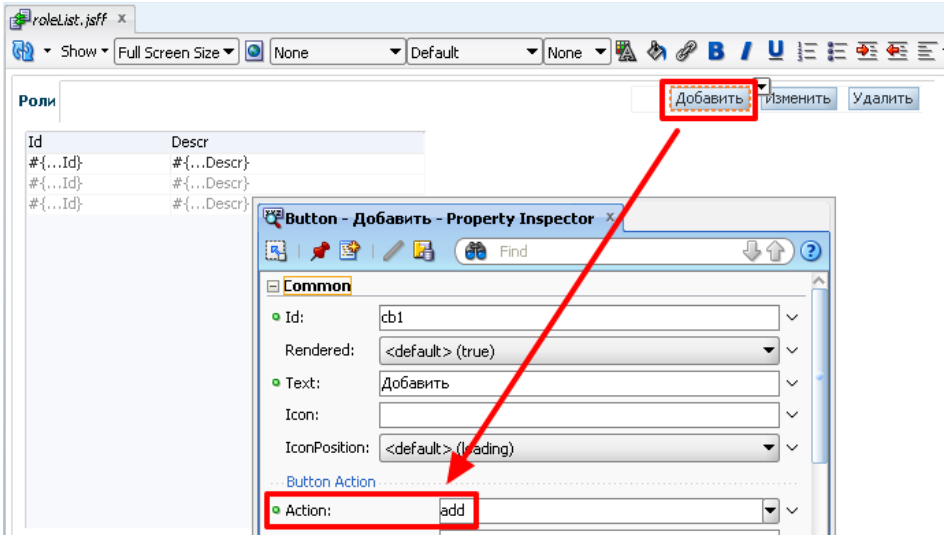


Настроим страницу roleList

На **PanelBox** разместим **3 элемента CommandButton**, и создадим таблицу для **SRoleView1**:

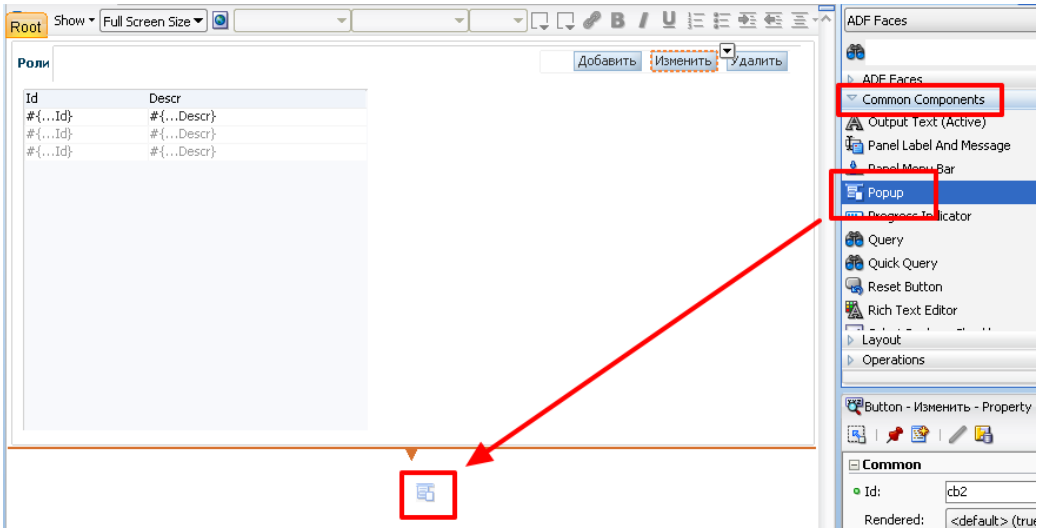


Зададим текст для кнопок, и настроим Action для кнопок «**Добавить**» (Action = **add**) и «**Изменить**» (Action = **edit**), а кнопку «**Удалить**» настроим дальше по-другому:

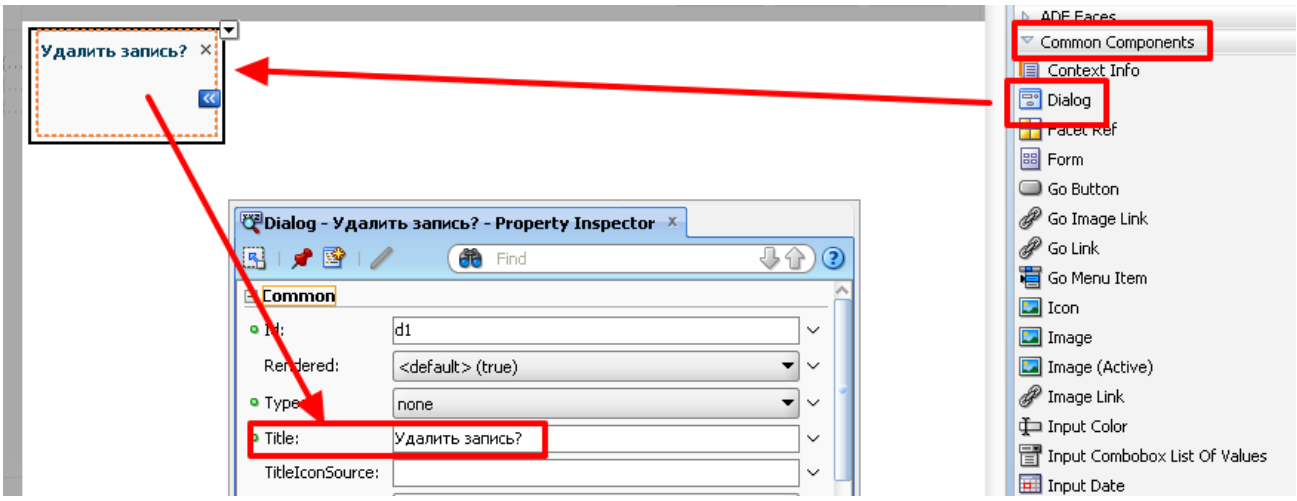


У кнопок «**Изменить**» и «**Удалить**» на свойстве **Disabled** так же стоит выражение:
#{bindings.SRoleView1.estimatedRowCount == 0}

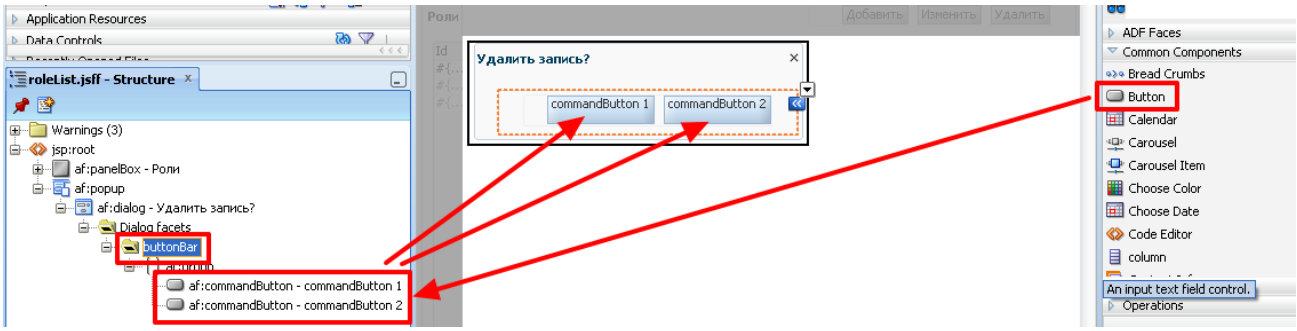
Скинем Popup и установим свойство **AutoCancel = enabled** (это лучше перепроверить, т.к. в JDeveloper 12c по умолчанию в этом свойстве выставлено **disabled**!):



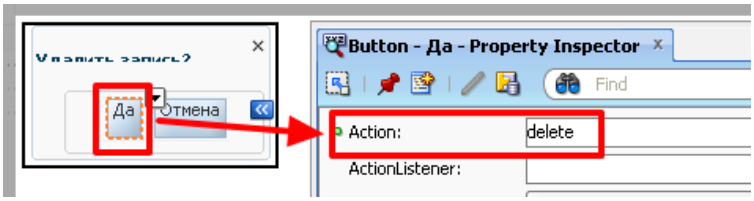
В Popup выложим диалоговое окно:



В **Dialog facets -> buttonBar** разместим 2 кнопки:



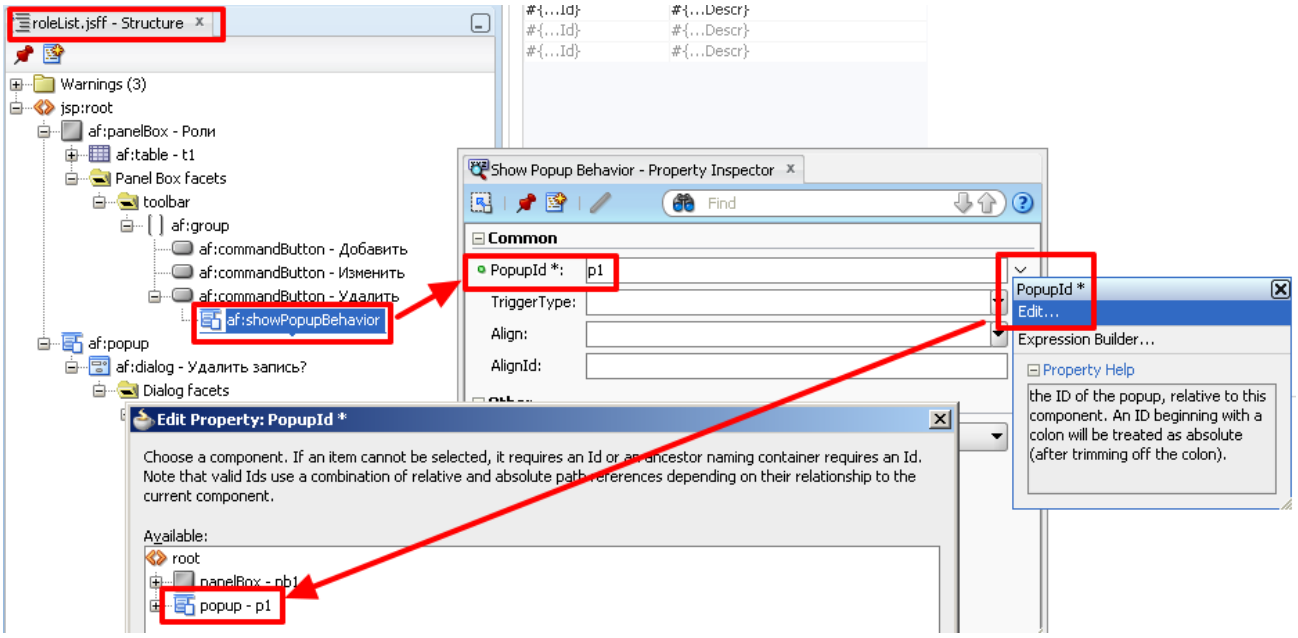
Зададим текст «**Да**» и «**Отмена**», и у кнопки «**Да**» зададим Action = **delete**:



У кнопок **PartialSubmit** должен равняться **false** (это лучше перепроверить, т.к. в JDeveloper 12c по умолчанию стоит **true**!).

Диалог для вопроса на удаление готов, переходим к настройки кнопки «**Удалить**». Скидываем прямо на нее следующий компонент:

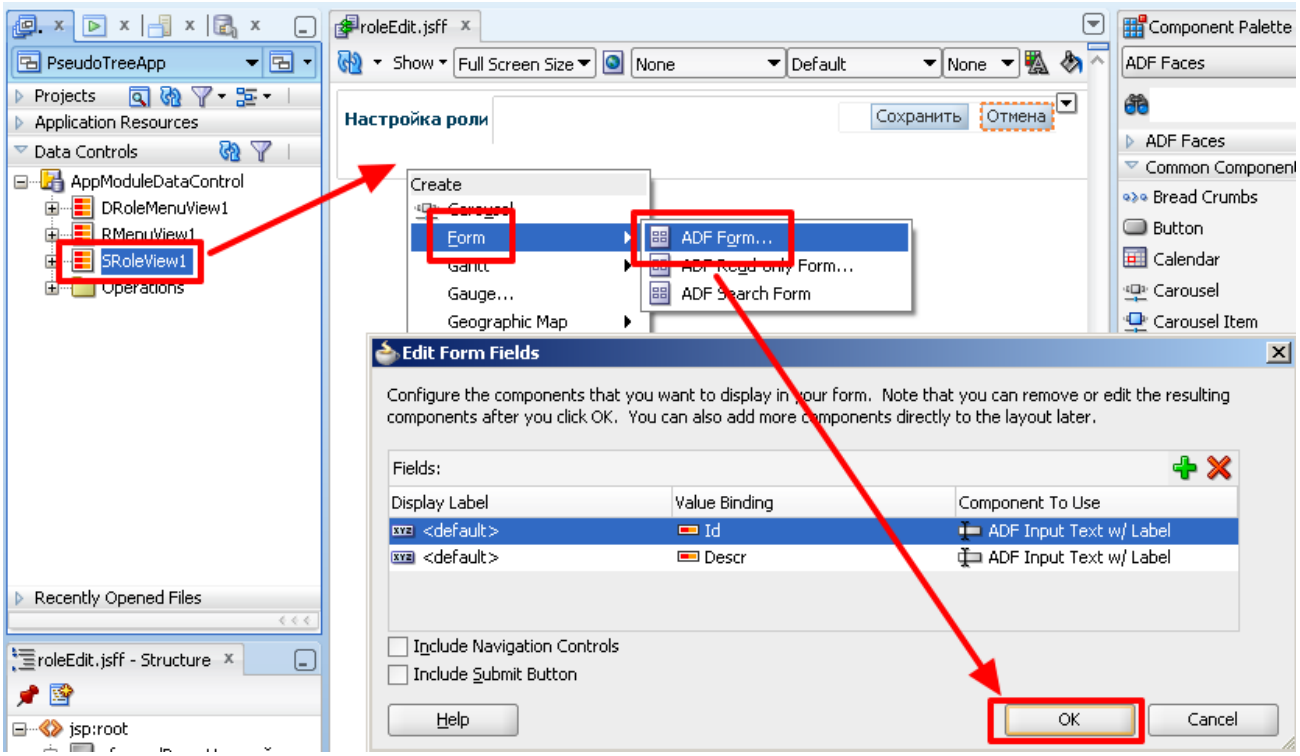
Укажем в свойстве PopupID идентификатор нашего Popup:



Все, кнопка «**Удалить**» настроена!

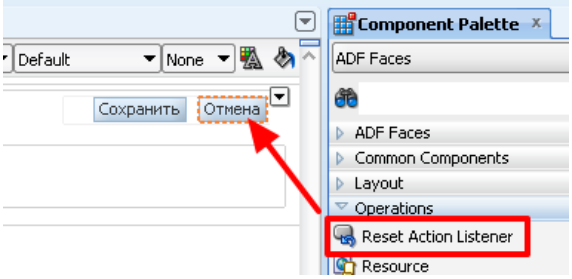
Настроим страницу roleEdit

На **PanelBox** разместим **2 элемента CommandButton**, и создадим форму для **SRoleView1**:



Зададим **Action** для кнопок «**Сохранить**» (Action = **save**) и «**Отмена**» (Action = **cancel**).

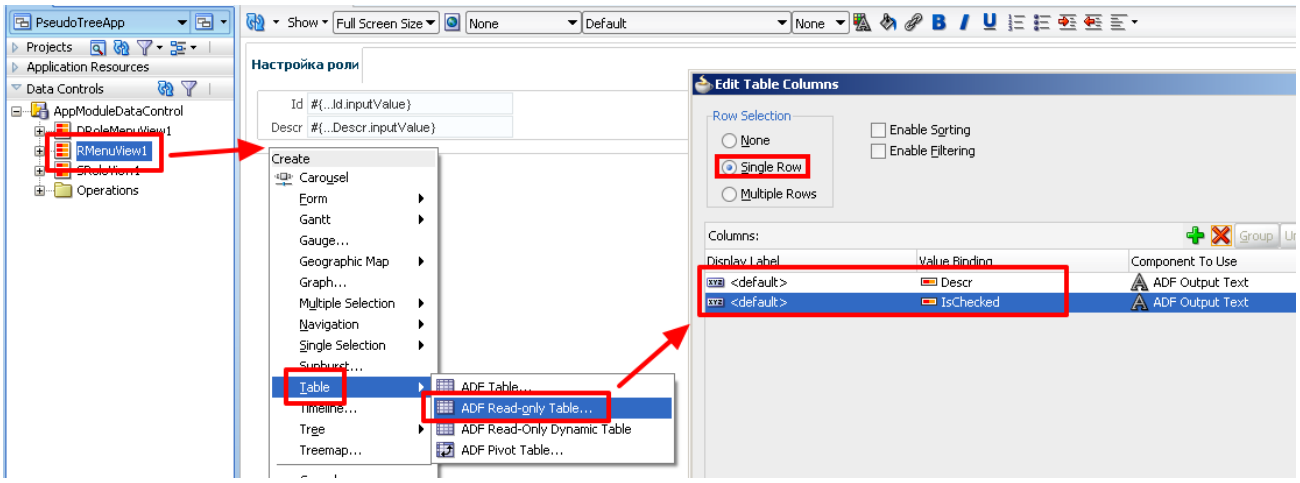
На кнопку «**Отмена**» дополнительно кинем следующий компонент (иначе, если этого не сделать, при Rollback может произойти разсинхронизация ViewObject, и курсор всегда будет оставаться на первой строке):



Так же устанавливаем свойство данной кнопки **Immediate = true**.

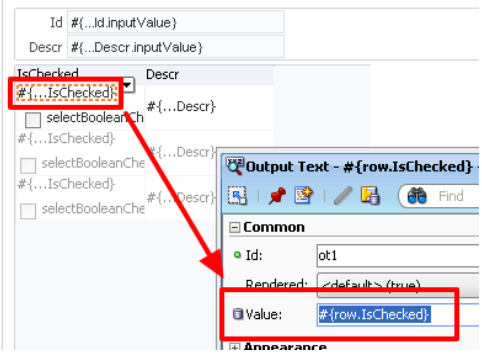
Наконец-то мы добрались до создания обещанного псевдо дерева

Скидываем View Object содержащий наше дерево (оставляем только **Descr** и **IsChecked**):

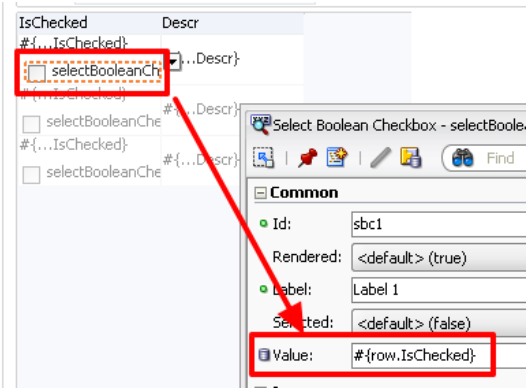


Скидываем обычный CheckBox:

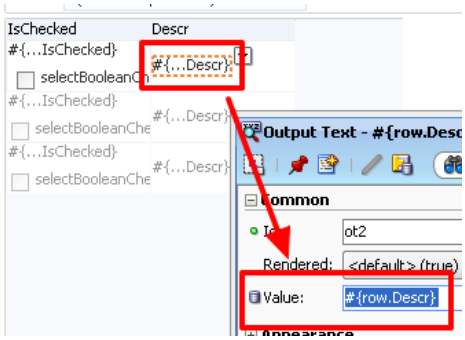
Копируем значение из свойства «Value» метки, которая была создана изначально:



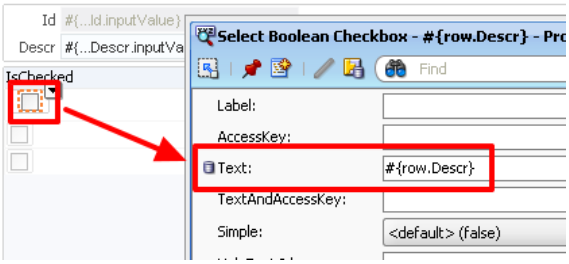
Вставляем это значение в свойство **Values** нашего CheckBox:



Таким же образом копируем значение свойства **Value** из поля **Descr**:



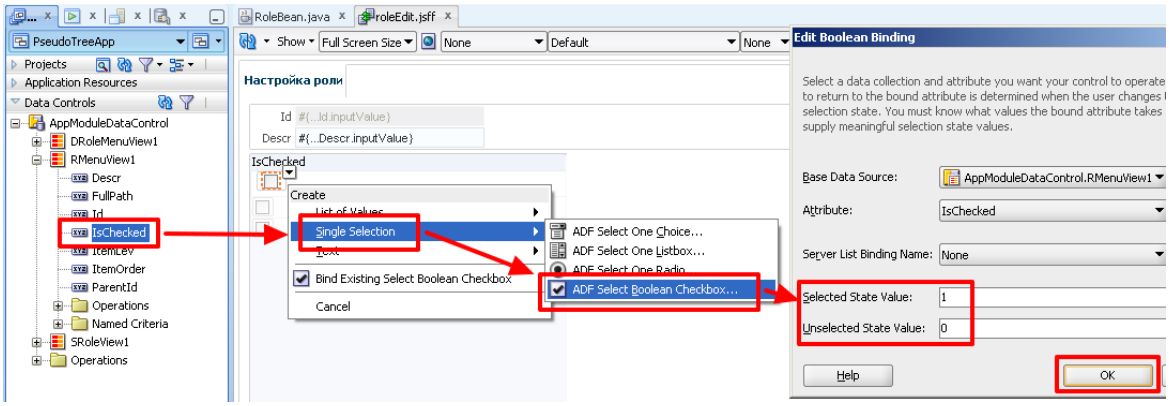
И вставляем его в свойство **Text** нашего CheckBox:



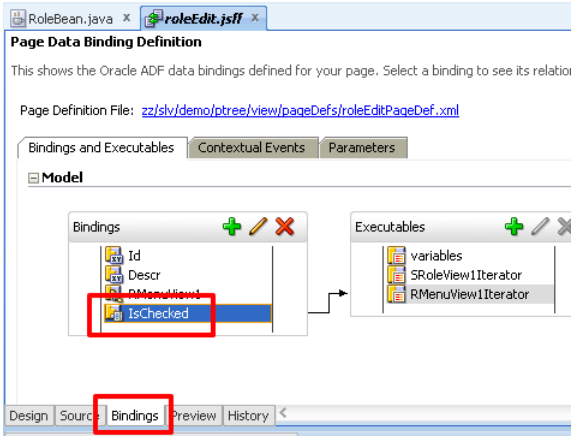
Дополнительно задаем у CheckBox **AutoSubmit=true** и **Immediate=true**.

Все, старую метку IsChecked можно удалить, а также удалить всю колонку Descr.

Для версии JDeveloper 11g, необходимо проделать следующее – перенесите еще один CheckBox в колонку таблицы, следующим образом:

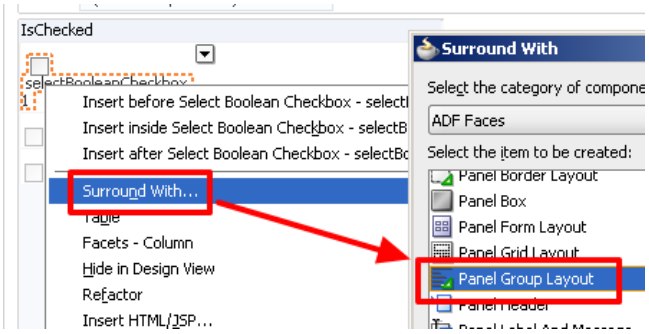


В результате создастся новый биндинг ссылающийся на тот же итератор:

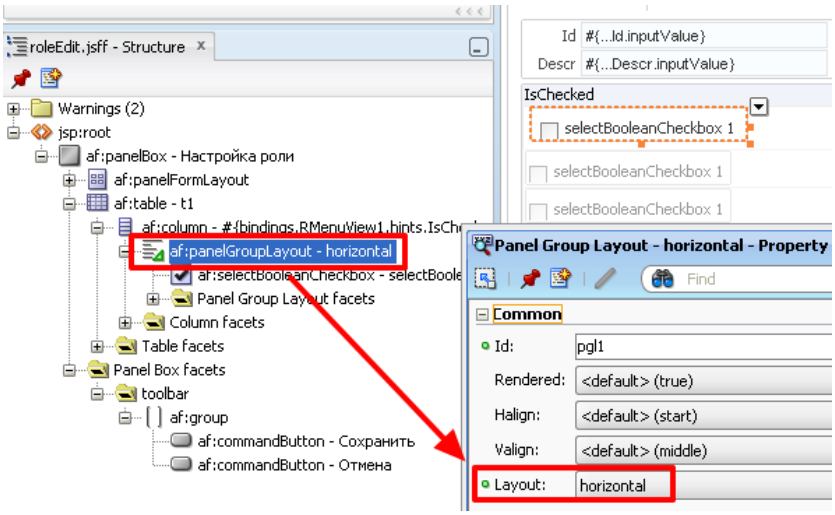


После этого этот второй чекбокс можно удалить.

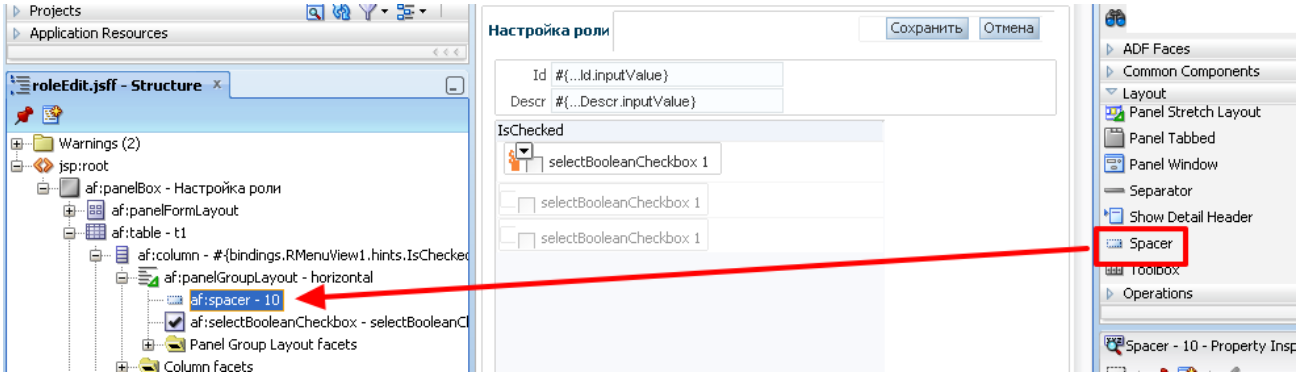
Идем, дальше. Обернем CheckBox используя Panel Group Layout:



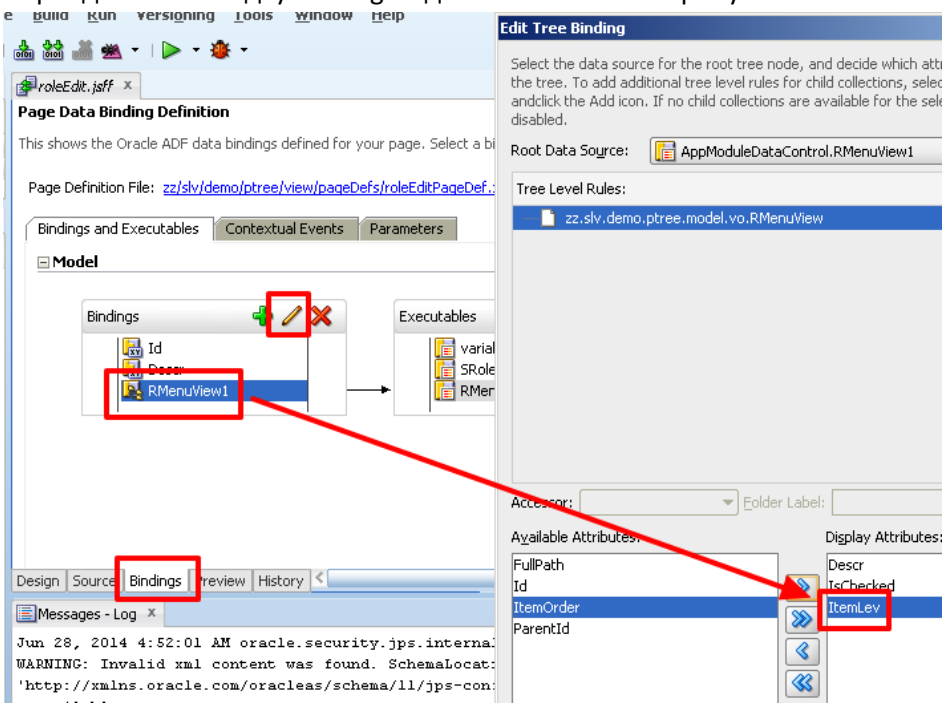
Установим вертикальное выравнивание



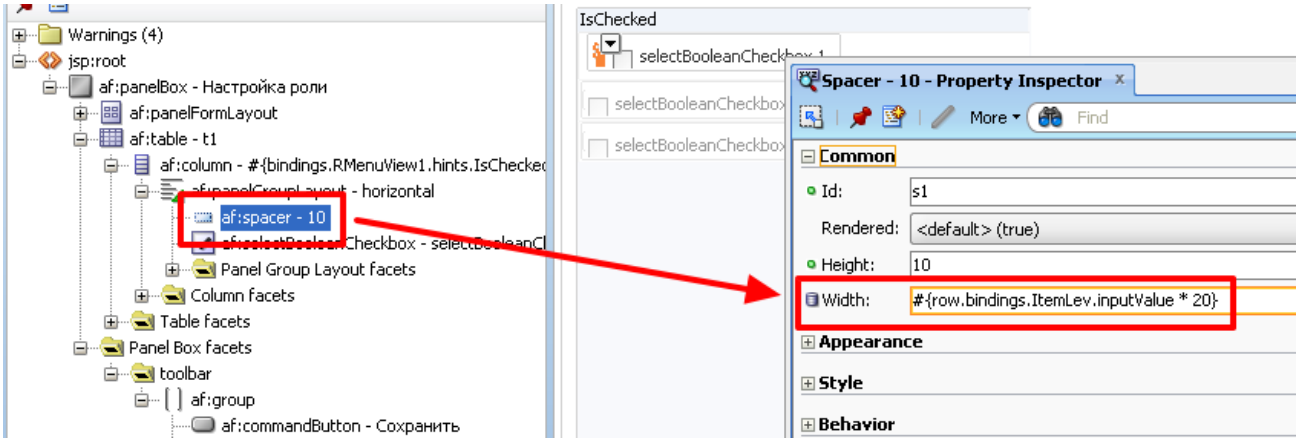
Добавим элемент **Spacer** перед CheckBox:



Перейдем на закладку Bindings и добавим в список атрибут **ItemLevel**:

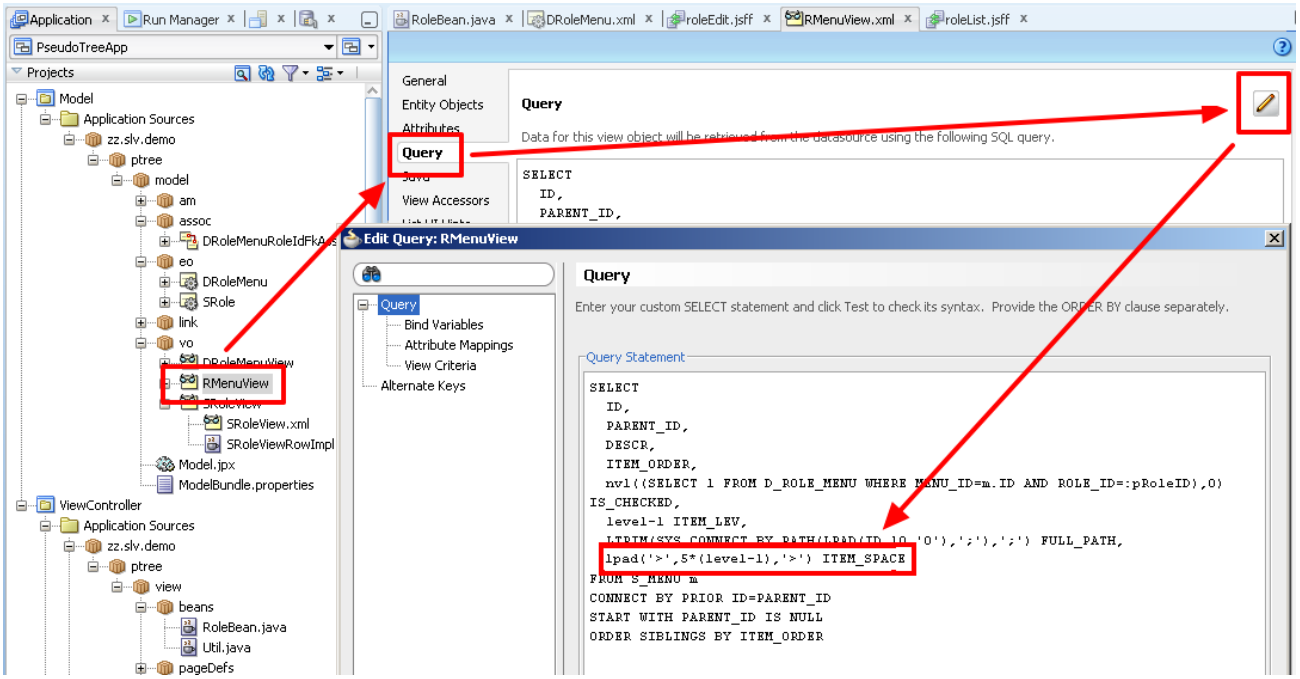


Теперь мы можем задать свойство **Width** для Spacer (т.е. его ширина будет вычисляться на основании значения атрибута **ItemLev**):

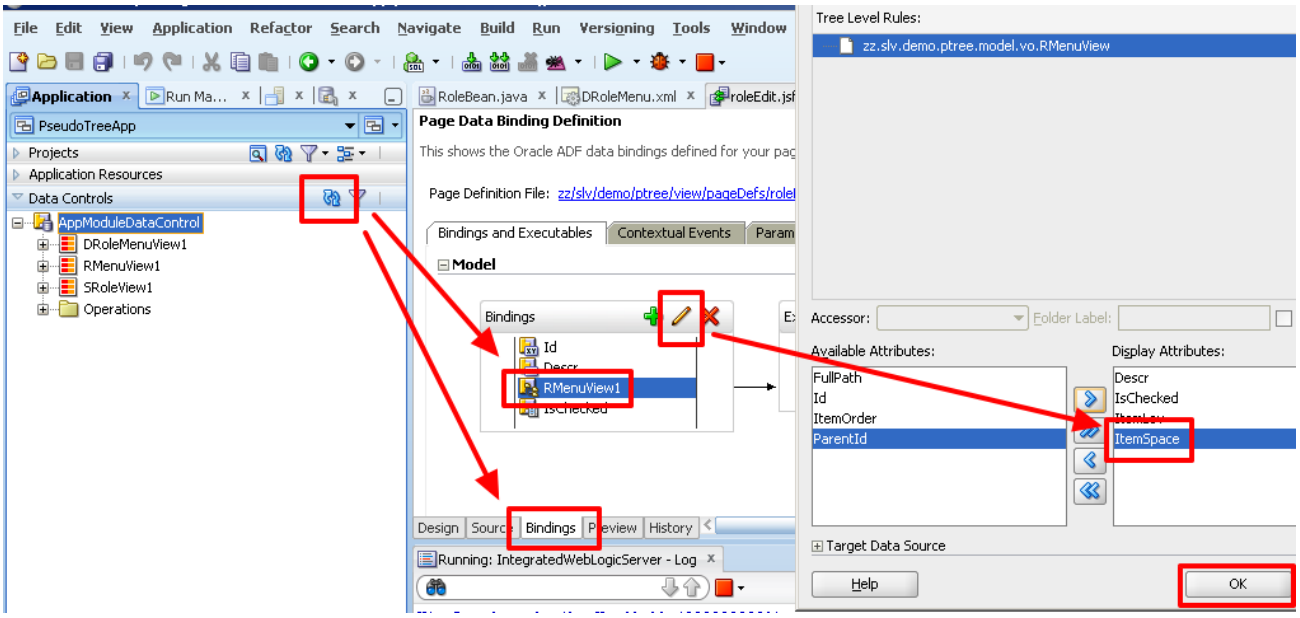


Увы, версия JDeveloper 11g, в отличии от 12c (там это отлично работает) не поняла такое выражение, поэтому Spacer удаляем, будем использовать обычную текстовую метку.

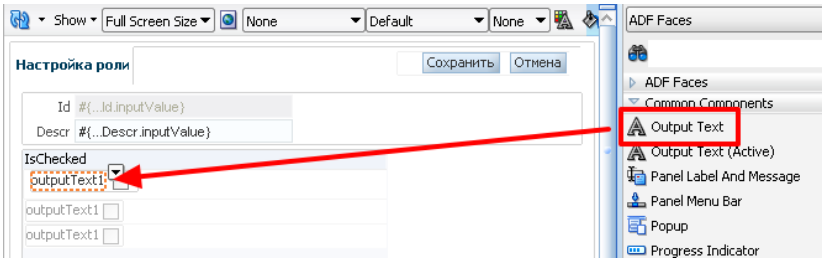
Первым делом дополним запрос полем **ITEM_SPACE** (увы пробелы OutputText схлопывает, поэтому будем изголяться):



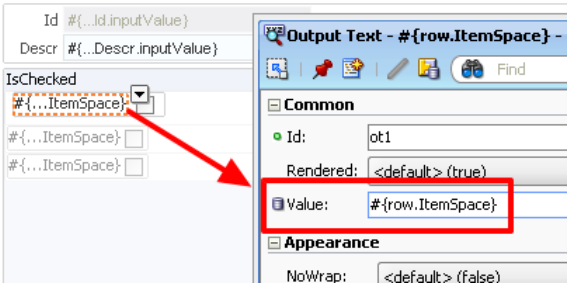
Обновляем **DataControl** и добавляем это поле **ItemSpace** в биндингах:



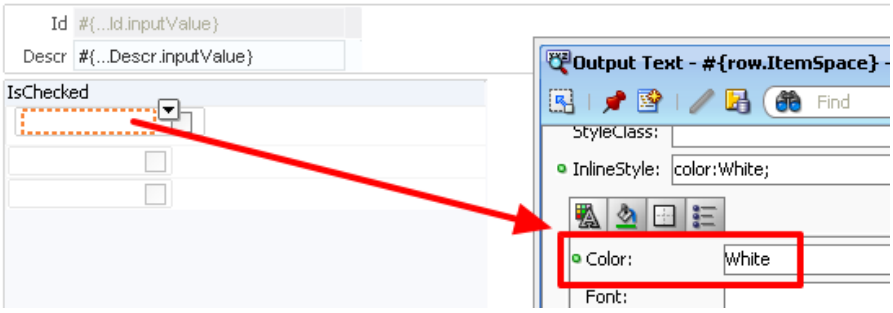
Кидаем вместо Spacer OutputText:



И задаем его свойство Value следующим образом (т.е. отступы будут делаться при помощи знаков «>»):

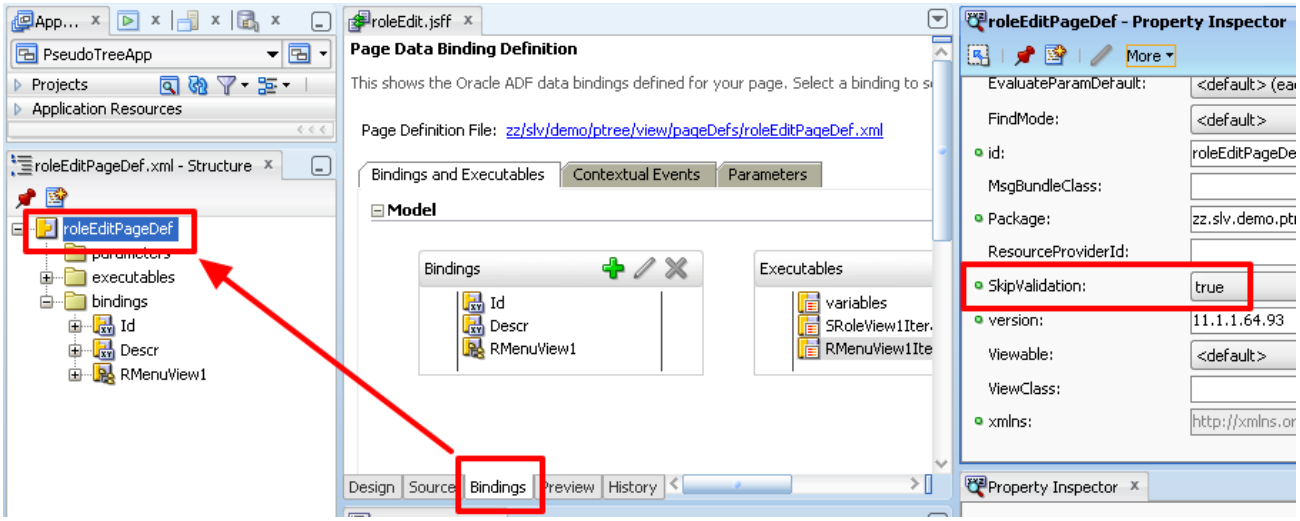


Пока придумал такой выход, покрасить текст в цвет фона, чтобы знаков «>» не было видно:



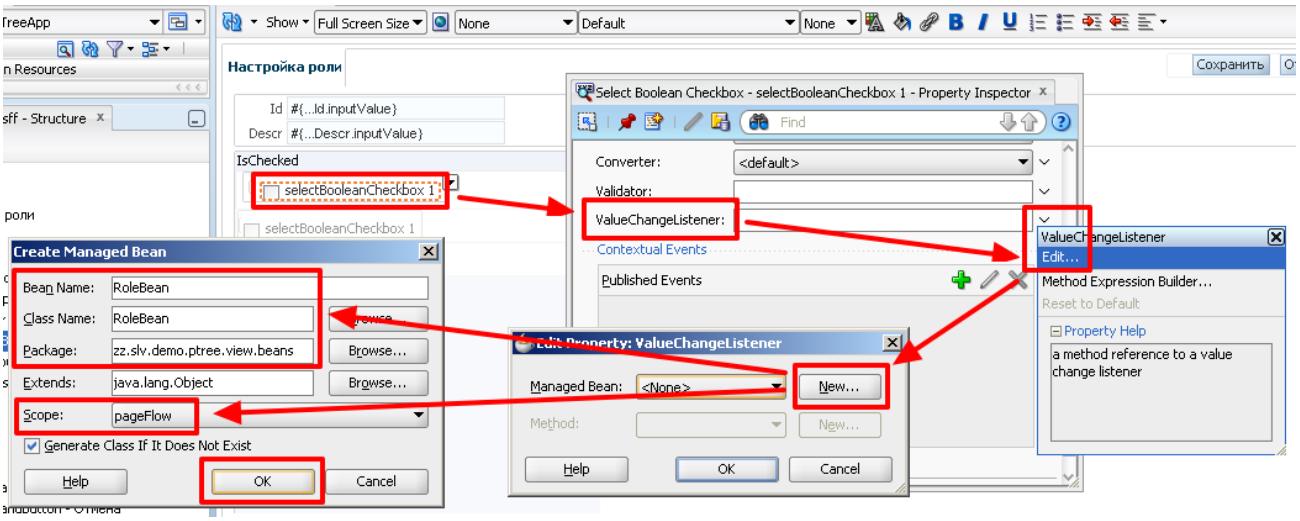
Идем дальше.

Так же дополнительно выставим у страницы **SkipValidation =** true (для того чтобы изменения в дереве не вызывали ошибку валидации для ролей):

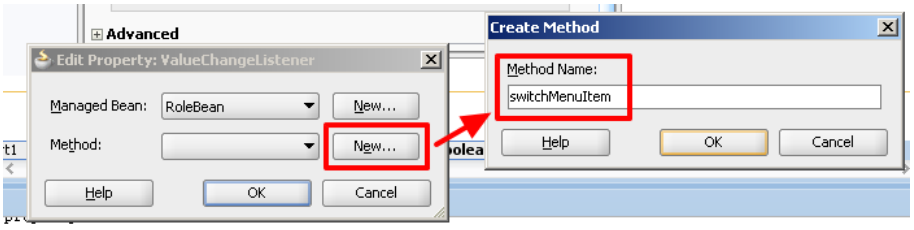


Настало время покодировать

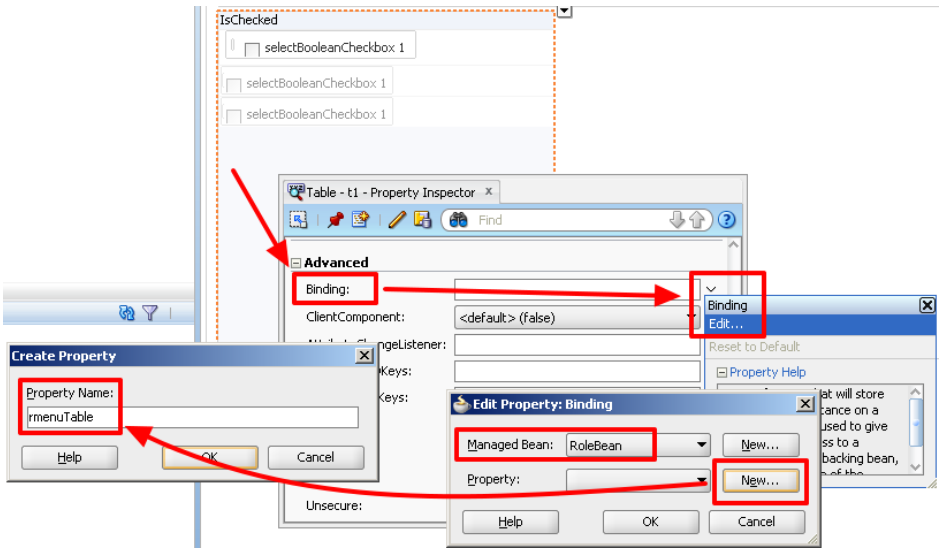
Определяем **ValueChangeListener** для CheckBox. Создаем бин:



Создаем метод:



Делаем привязку таблицы к свойству бина:



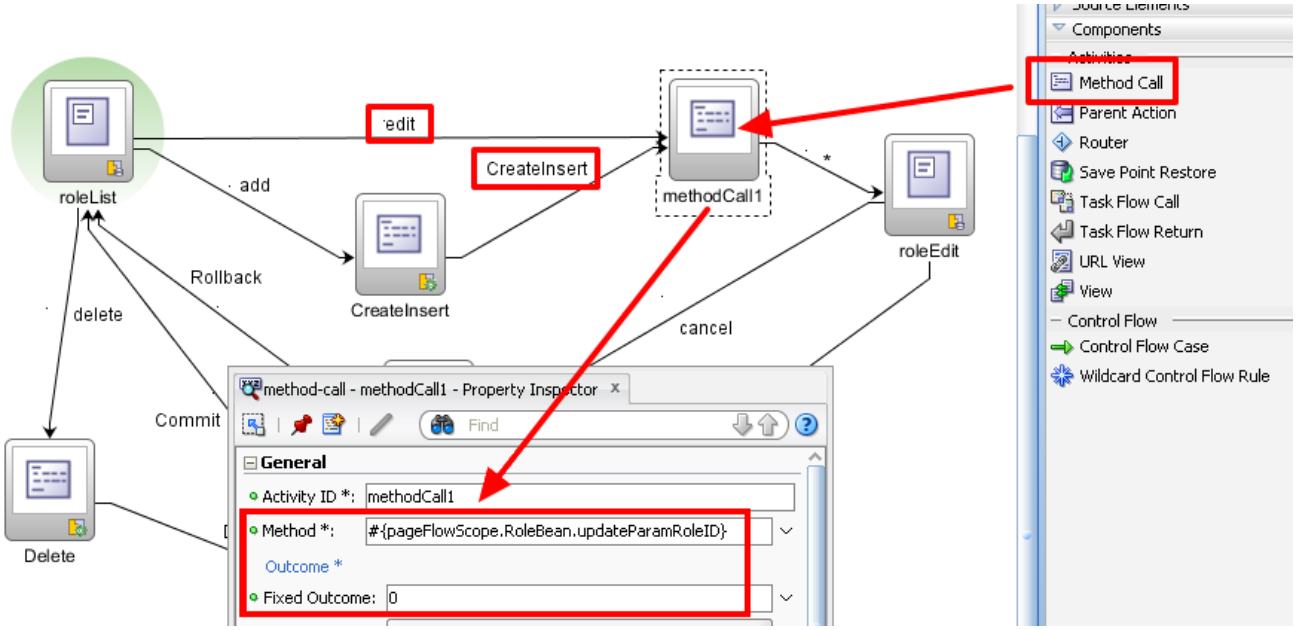
Дальше необходимо изучать код класса «**PseudoTreeApp\ViewController\src\zz\slv\demo\ptree\view\beans\RoleBean.java**».

Общая логика следующая, при изменении значения CheckBox в строке, мы обновляем значения дочерних и родительских строк в **RMenuView1**, параллельно удаляя или добавляя значения в подчиненный **DRoleMenuView1** (он связан при помощи линка с **SRoleView1**).

В данном классе, так же, есть метод, который отвечает за обновления View Object отвечающего за вывод меню:

```
public void updateParamRoleID() {
    System.out.println("START");

    try {
        Row roleRow = roleVO.getCurrentRow();
        roleID = Integer.parseInt(roleRow.getAttribute("Id").toString());

        System.out.println("Role ID = " + roleID);

        VariableValueManager vvm = rmenuVO.ensureVariableManager();
        vvm.setVariableValue("pRoleID", roleID);
        rmenuVO.executeQuery();

        System.out.println("Parameter pRoleID is Updated");

    } catch (Exception e) {
        System.out.println(e);
    }

    System.out.println("FINISH");
}
```

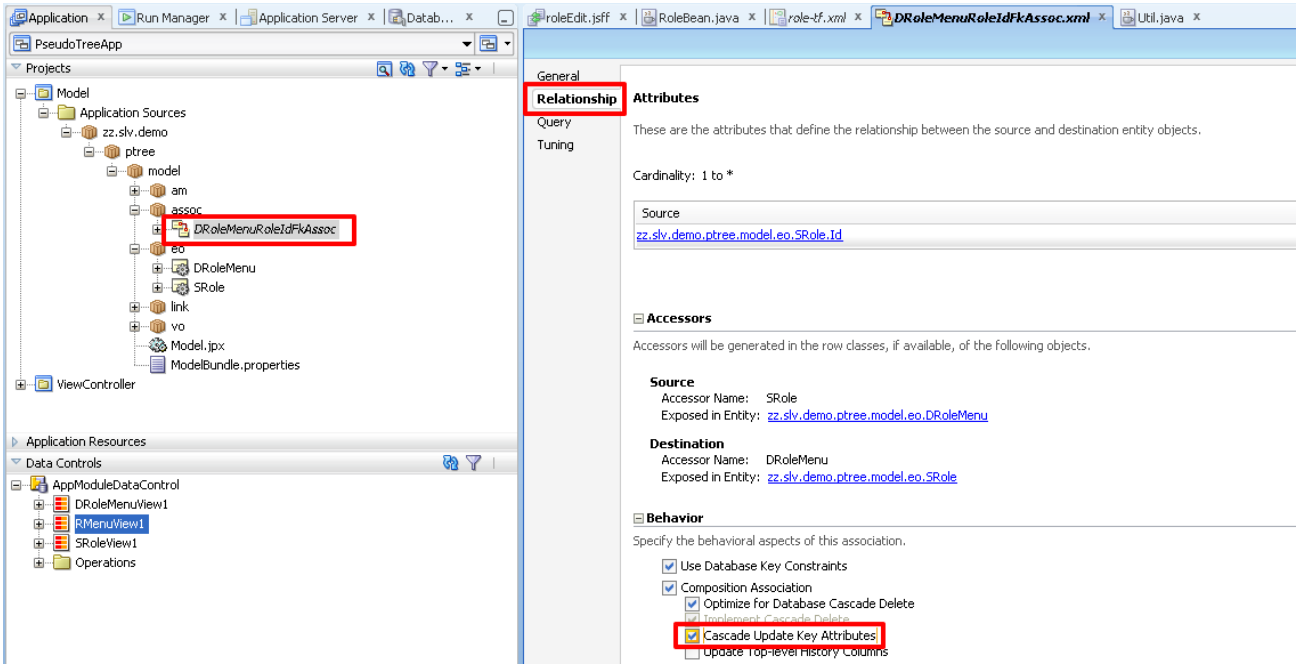
Вызов этого метода мы добавим в наш **TaskFlow**, следующим образом:



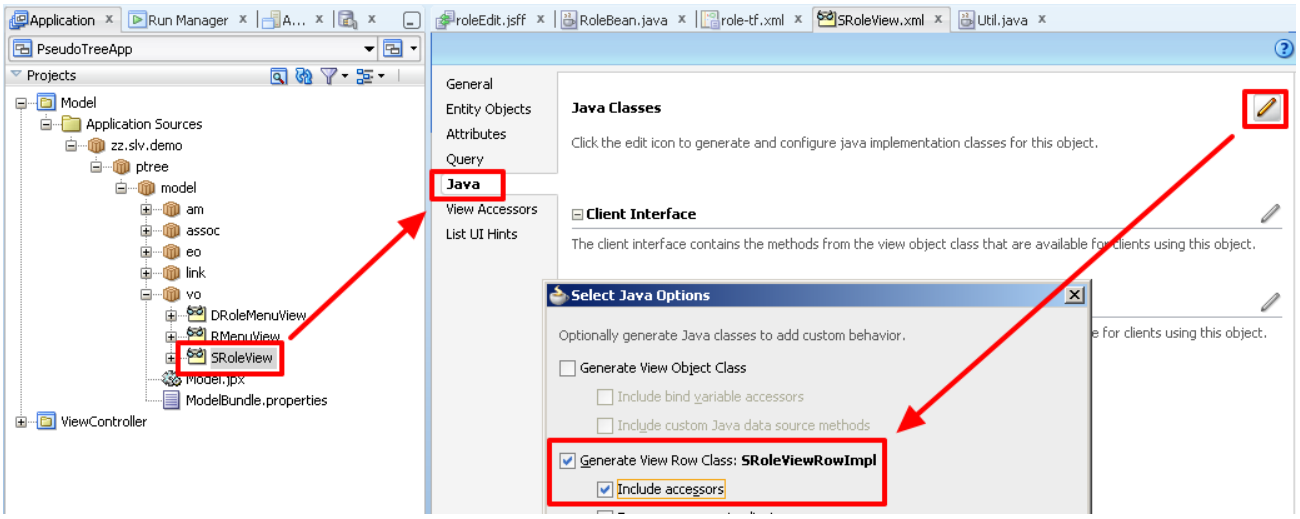
Обходим ошибку InvalidOwnerException JBO-25030

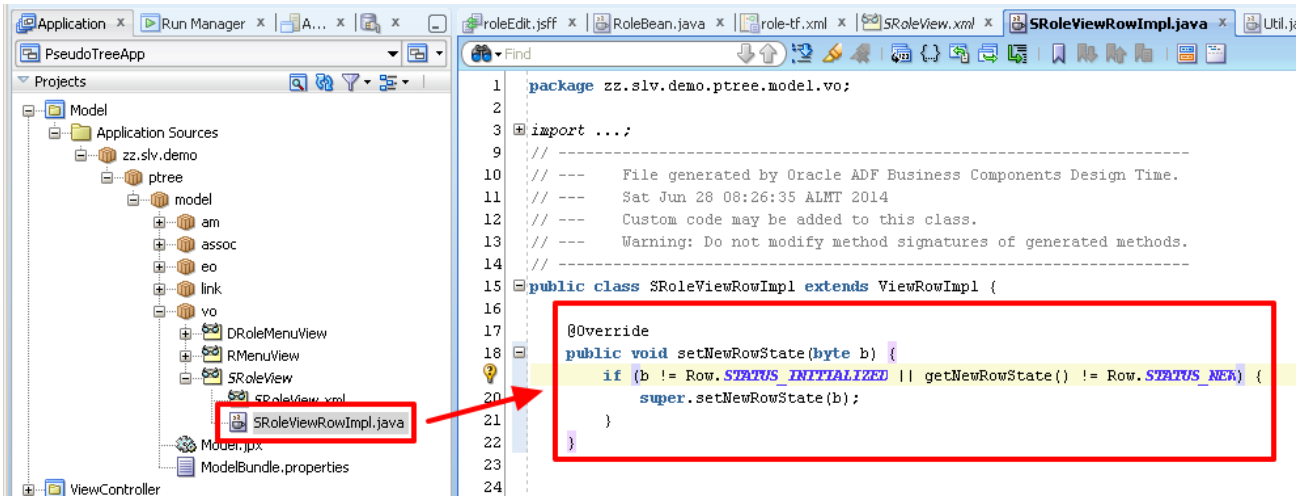
Данная ошибка возникает при одновременном создании master- и detail-записей.

Решение взято с сайта - http://stan1slav.blogspot.com/2014/05/adf-invalidownerexception-jbo-25030.html



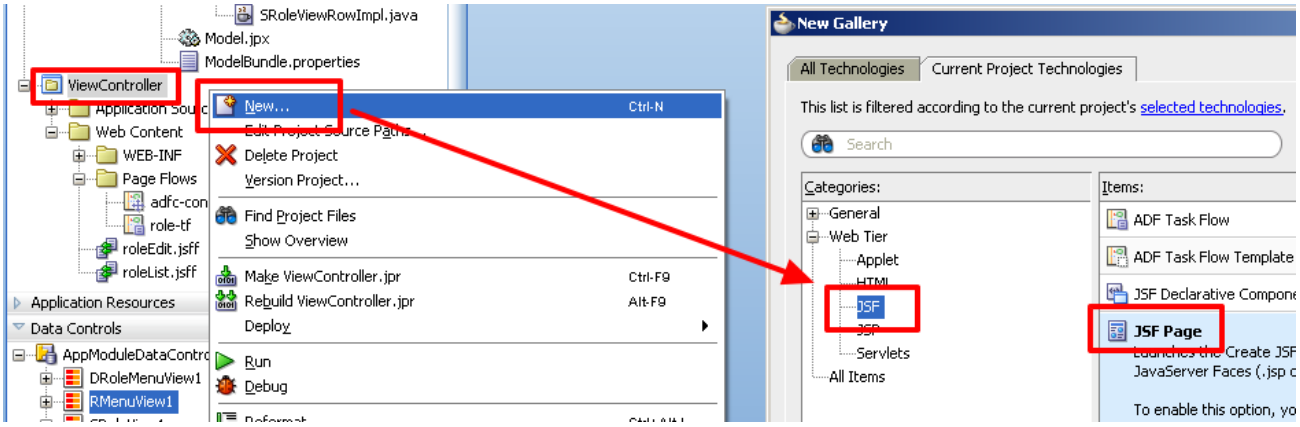
В главном View Object:

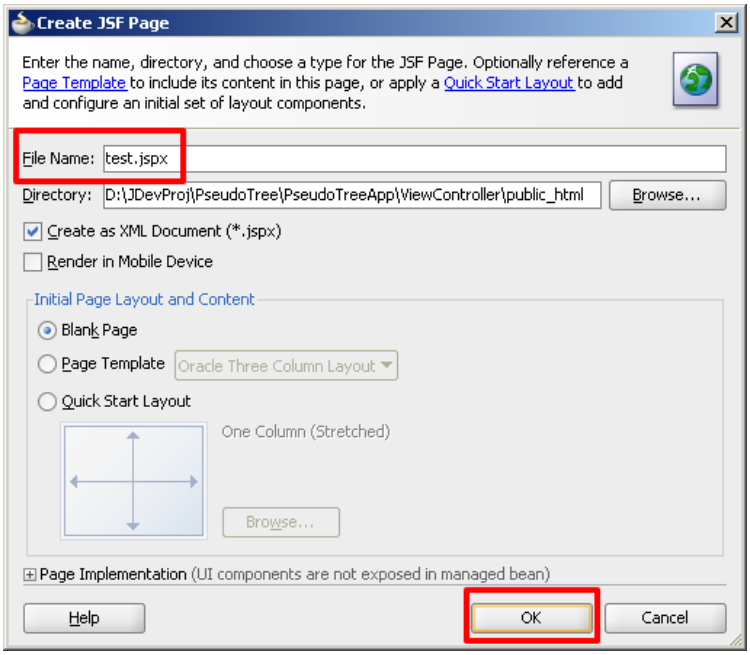




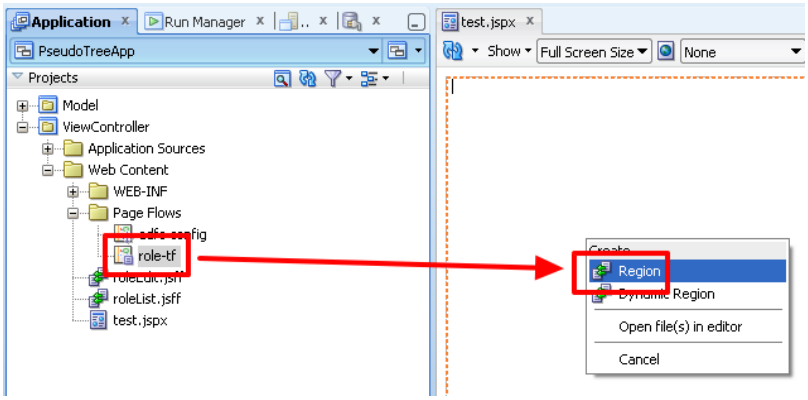
Тестируем проект

Создадим для тестов страницу:

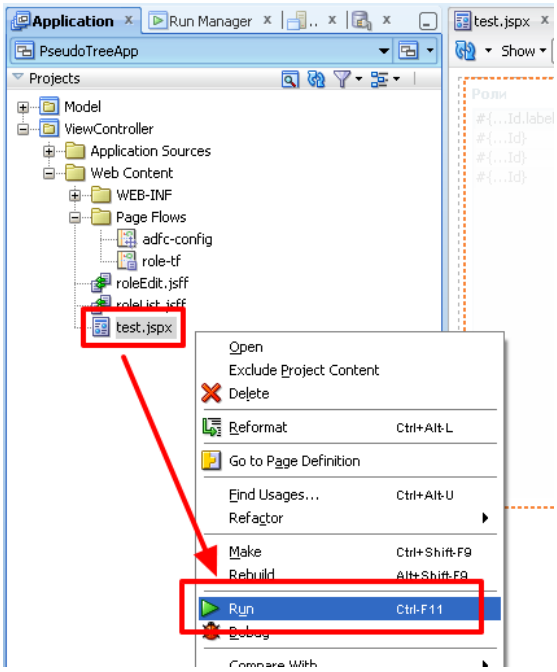




На нее скинем наш TaskFlow:



Все готово, запускаем проект:



Вот и все, о чем я хотел сегодня рассказать – дерево получилось примитивным (даже не умеет сворачиваться), но зато вполне предсказуемым.

Удачного вам изучения ADF!

PS. Видели ночь, писали всю ночь до утра…