Расчет заработной платы: База данных-1

МКД 03.01 «Технология разработки программного обеспечения»

0. Эпиграф

Часто у экспертов данных больше,
чем здравого смысла.

Колин Пауэлл

1. Вопросы реализации базы данных

Теперь, когда большая часть приложения проанализирована, спроектирована и реализована, можно вернуться к вопросу о базе данных.

Ясно, что класс **БазаДанных** (**PayrollDatabase**) инкапсулирует идею постоянного хранения. Объекты, находящиеся в **БазеДанных** (**PayrollDatabase**), очевидно, не должны уничтожаться после завершения работы приложения. Как это реализовать?

Есть несколько вариантов.

2. Вопросы реализации базы данных

Можно реализовать **БазуДанных** (**PayrollDatabase**), воспользовавшись какой-нибудь объектно-ориентированной системой управления базами данных (ОО-СУБД). Это дало бы возможность помещать объекты на постоянное хранение в базу. Нам как проектировщикам не пришлось бы прилагать особых усилий, потому что ООСУБД почти ничего не добавила бы к дизайну.

Одно из серьезных достоинств ООСУБД заключается в том, что они мало влияют на объектную модель приложения. С точки зрения дизайна, базы данных как бы и не существует.

3. Вопросы реализации базы данных

Другой вариант – воспользоваться для хранения данных плоскими файлами. На этапе инициализации объект **БазаДанных** (**PayrollDatabase**) мог бы прочитать файл и создать необходимые объекты в памяти. А в конце работы он просто записал бы новую версию файла.

Конечно, это решение не подойдет, если в компании сотни или тысячи работников или если необходим оперативный одновременный доступ к базе данных о работниках.

Однако для небольшой компании его может оказаться достаточно, и уж точно оно годится как механизм тестирования всех остальных классов приложения без инвестирования в большую СУБД.

4. Вопросы реализации базы данных

Третий вариант – подключить к объекту **БазаДанных** (**PayrollDatabase**) реляционную СУБД (РСУБД). Тогда реализация **БазаДанных** (**PayrollDatabase**) свелась бы к выполнению запросов к СУБД для временного создания необходимых объектов в памяти.

Важно, что любой из этих подходов будет работать. Мы спроектировали приложение так, что оно не знает и не интересуется механизмом реализации базы данных. С точки зрения приложения, база данных – это просто средство для управления постоянным хранением.

5. Вопросы реализации базы данных

Обычно базы данных не следует рассматривать как решающий фактор при проектировании и реализации. Как мы только что показали, этот вопрос можно отложить до последнего момента и трактовать базу данных как деталь реализации. При этом мы оставляем открытым ряд интересных возможностей для реализации постоянного хранения и создания механизмов тестирования остальных частей приложения.

Мы также не связываем себя с конкретной технологией баз данных или продуктом. Мы свободны выбрать потребную базу данных, исходя из получившегося дизайна, а впоследствии заменить один продукт на другой.

6. Вопросы реализации базы данных

На множестве предыдущих занятий мы реализовали всю бизнес-логику системы расчета зарплаты. В состав реализации входит и класс **БазаДанных** (**PayrollDatabase**), предназначенный для хранения необходимых данных в памяти.

В тот момент его было вполне достаточно.

Но ясно, что системе необходимо какое-то более долговременное хранилище данных.

Рассмотрим, как обеспечить сохранение данных в реляционной базе.

7. Выбор СУБД

Выбор технологии баз данных обычно диктуется скорее политическими, нежели техническими причинами. Компании-производители баз данных и платформ немало постарались, чтобы убедить рынок в критической важности такого выбора. Лояльность и преданность конкретному поставщику формируется по причинам, больше относящимся к человеческим отношениям, чем к технической целесообразности.

Поэтому не следует придавать слишком большого значения нашему выбору Microsoft SQL Server в качестве СУБД для хранения данных приложения.

8. Схема БД

На следующем слайде показана схема базы данных, которой мы будем далее пользоваться.

В центре находится таблица **Работник** (**Employee**). Здесь хранятся данные о работниках, а также строковые константы, определяющие график выплат **ТипГрафика** (**ScheduleType**), метод платежа **ТипМетода** (**PaymentMethodType**) и тарификацию **ТипВыплат** (**PaymentClassifica-tion**).

9. Схема БД

Сами данные, относящиеся к тарификации, хранятся в таблицах **ВыплатаЧас** (**HourlyClassification**), **ВыплатаОклад** (**SalariedClassification**) и **ВыплатаКомис** (**CommissionedClassification**).

Ссылка на соответствующую запись таблицы **Работник** (**Employee**) находится в столбце **РабНом** (**EmpId**). Для этого столбца определено ограничение, гарантирующее, что в таблице **Работник** (**Employee**) существует запись с данным значением **РабНом** (**EmpId**).

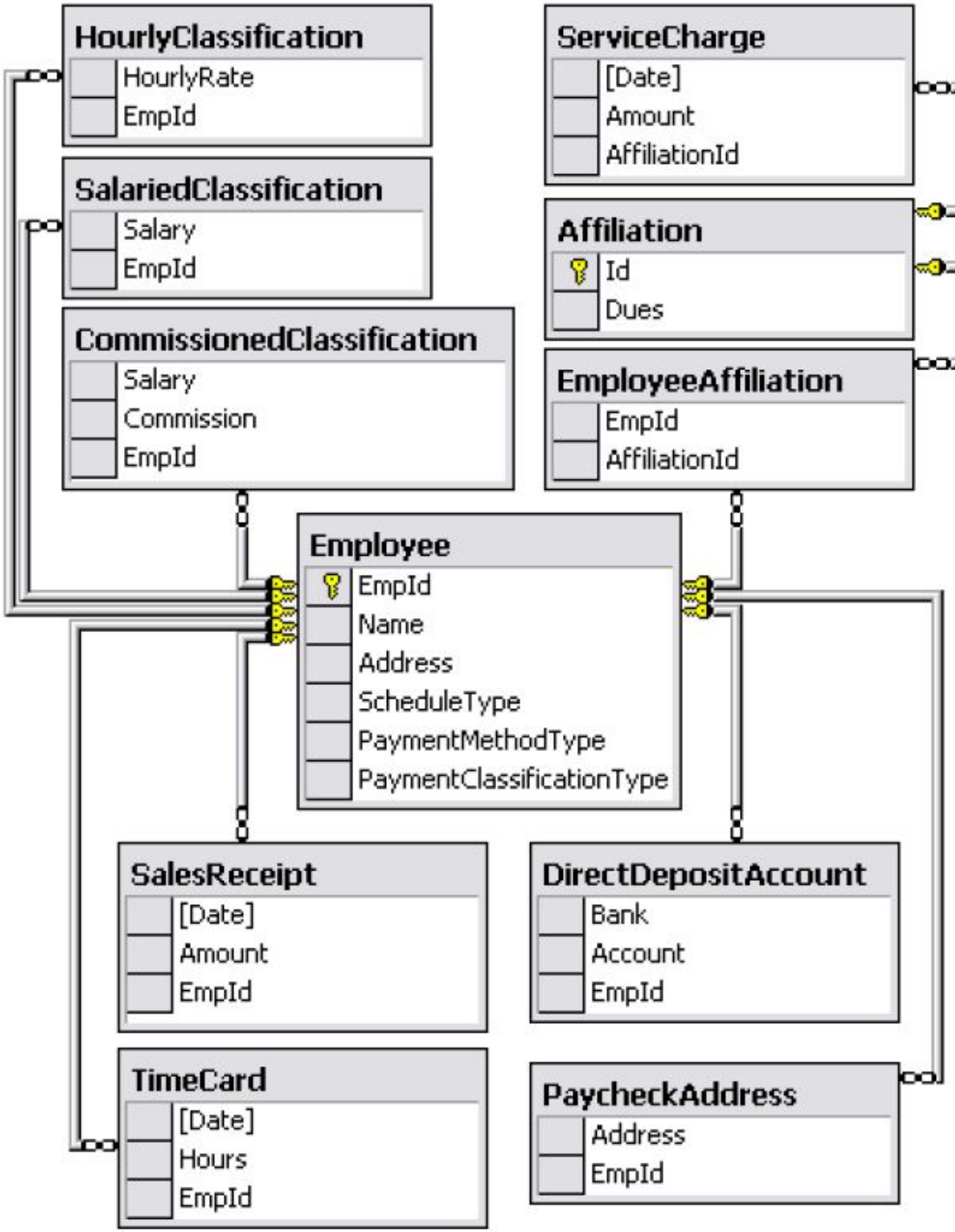
10. Схема БД

В таблицах **DirectDepositAccount** () и **PaycheckAddress** () хранятся данные, относящиеся к соответствующему методу (перечисление на банковский счет или отправка чека почтой); на столбец **РабНом** (**EmpId**) наложено аналогичное ограничение.

Таблицы **SalesReceipt** (**СправкиПродаж**) и **TimeCard** (**КартыТабУчета**) не нуждаются в пояснениях.

В таблице Affiliation хранятся данные о членстве в профсоюзе, она связана с таблицей Employee через таблицу EmpoyeeAffiliation.

11. Схема БД


HourlyClassification
HourlyRate
EmpId
SalariedClassification
Salary
EmpId
CommissionedClassification
Salary
Commission
EmpId
ServiceCharge
[Date]
Amount
AffiliationId
Affiliation
Id
Dues
EmployeeAffiliation
EmpId
AffiliationId
Employee
EmpId
Name
Address
ScheduleType
PaymentMethodType
PaymentClassificationType
SalesReceipt
[Date]
Amount
EmpId
TimeCard
[Date]
Hours
EmpId
DirectDepositAccount
Bank
Account
EmpId
PaycheckAddress
Address
EmpId

12. Создание БД

В Обозревателе решений выделите ПРОЕКТ Payroll_FIO (иначе база добавится в текущую папку!)

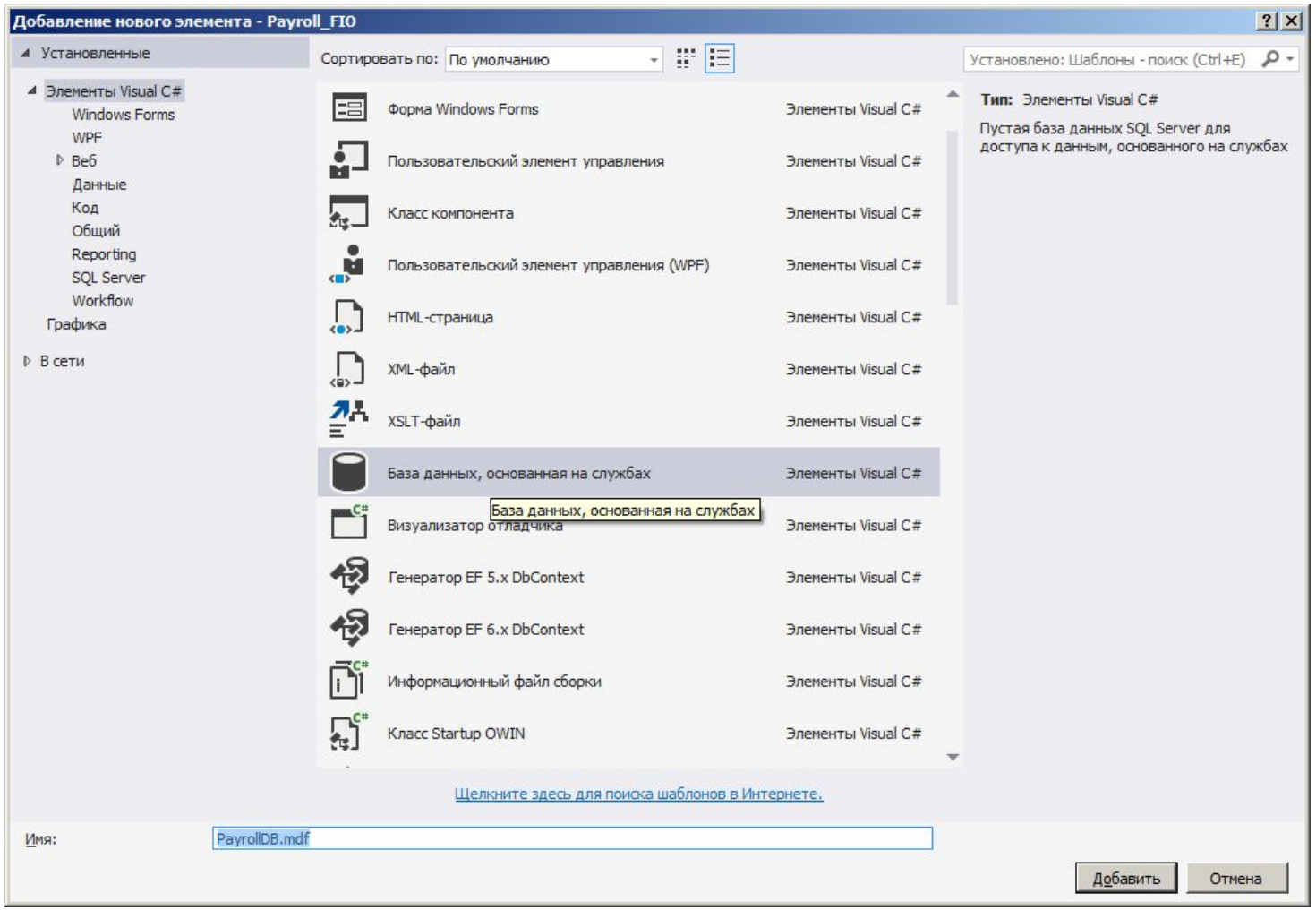
В главном меню выберите ***Проект*** -> ***Добавление нового элемента***.

Откроется диалоговое окно ***Добавить новый элемент***, в котором можно добавлять элементы, допустимые в проекте Windows Forms.

В списке шаблонов элементов прокрутите вниз до тех пор, пока не появится ***База данных, основанная на службах***, и выберите его.

Назовите базу данных **PayrollDB.mdf** и нажмите кнопку ***Добавить***.

13. Добавление БД в проект



14. Создание БД

Если окно "*Источники данных*" еще не открыто, нажмите клавиши SHIFT+ALT+D или выберите в строке меню ***Вид -> Другие окна -> Источники данных***.

В окне "*Источники данных*" щелкните ссылку ***Добавить новый источник данных***.

В окне ***Мастер настройки источника*** данных нажмите кнопку ***Далее*** четыре раза, чтобы принять параметры по умолчанию, затем кнопку ***Готово***. **ПРОЧИТАЙТЕ С ЧЕМ СОГЛАШАЕТЕСЬ!!!** Обратите внимание, что VS сама находит созданную вами БД, сама к ней подключается.

В окне свойств базы данных отображается строка подключения и расположение основного MDF-файла базы данных.

15. Создание таблиц

- Создайте таблицы в соответствии со схемой данных. (см. слайды далее!)
- Не забудьте указать первичный ключ в каждой таблице.
- Создайте внешние ключ, чтобы определить, как записи в одной таблице могут соотноситься с записями в другой таблице.

16. Создание таблицы Работник (Employee)

Чтобы создать таблицу Работник (Employee)

В области ***Обозреватель серверов*** разверните узел ***Подключения данных***, а затем узел **PayrollDB.mdf**.

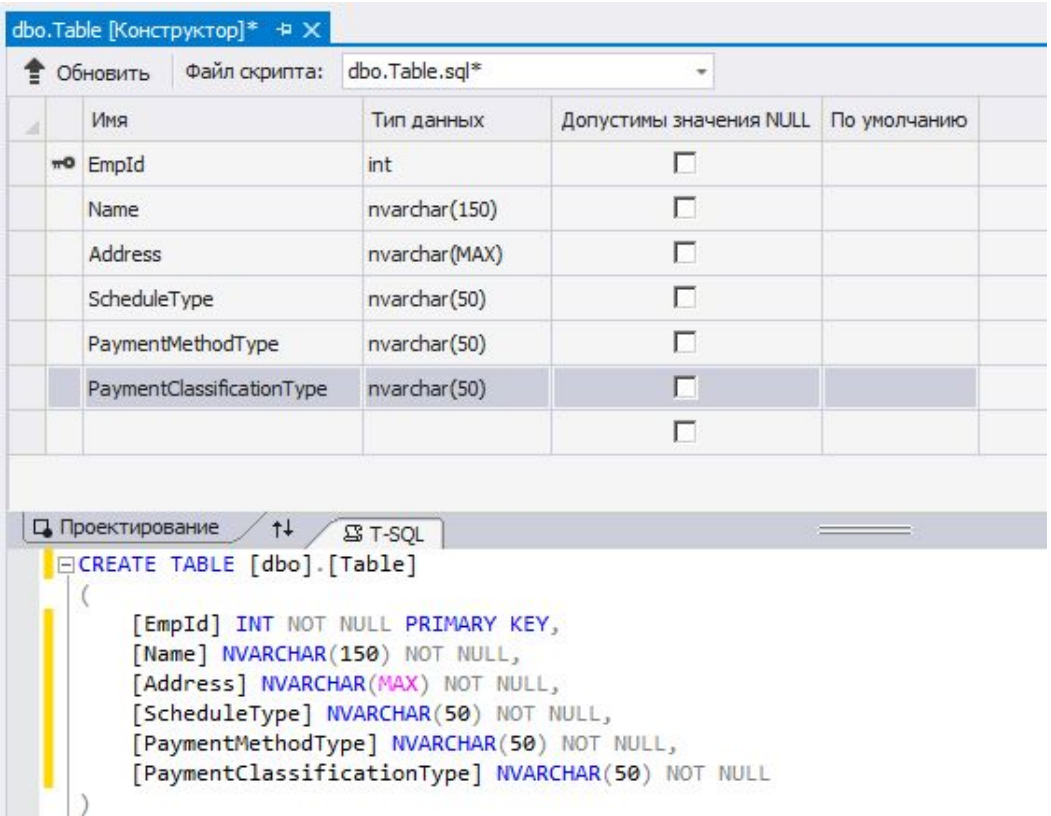
Если обозреватель еще не открыт, выберите в строке меню ***Вид -> Обозреватель серверов***.

Откройте ***контекстное меню*** для раздела ***Таблицы***, выберите ***Добавить новую таблицу***.

Будет открыт ***Конструктор таблиц***, отобразится сетка с одной строкой по умолчанию, которая представляет один столбец в создаваемой таблице. Путем добавления строк в сетку будут добавлены столбцы в таблицу.

17. Создание таблицы Работник (Employee)

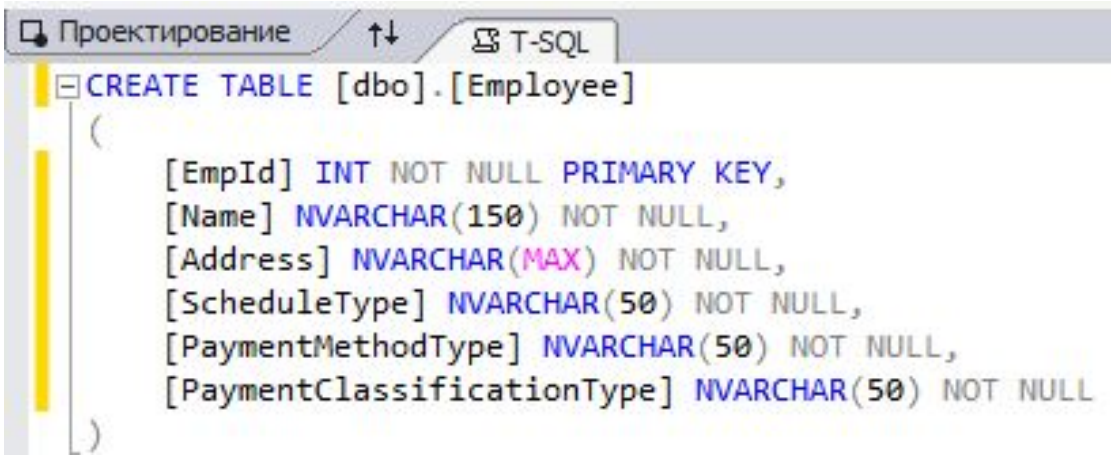
В сетке добавьте строку для каждой из следующих записей, указав имя столбца, тип данных и допустимы значения null:



18. Создание таблицы Работник (Employee)

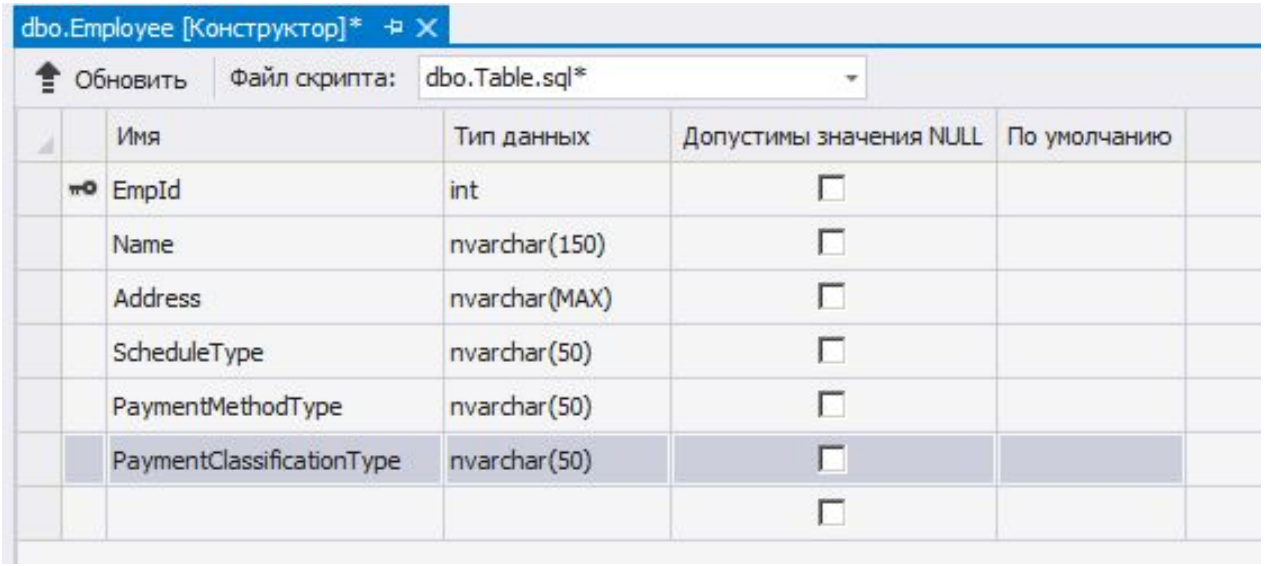
Откройте контекстное меню для столбца **EmpID** и выберите ***Задать первичный ключ***. (НЕ ВЫБЕРИТЕ случайно ***УДАЛИТЬ первичный ключ***!)

Назовите таблицу “**Employee**" путем обновления первой строки в области скриптов, как показано ниже:



19. Создание таблицы Работник (Employee)

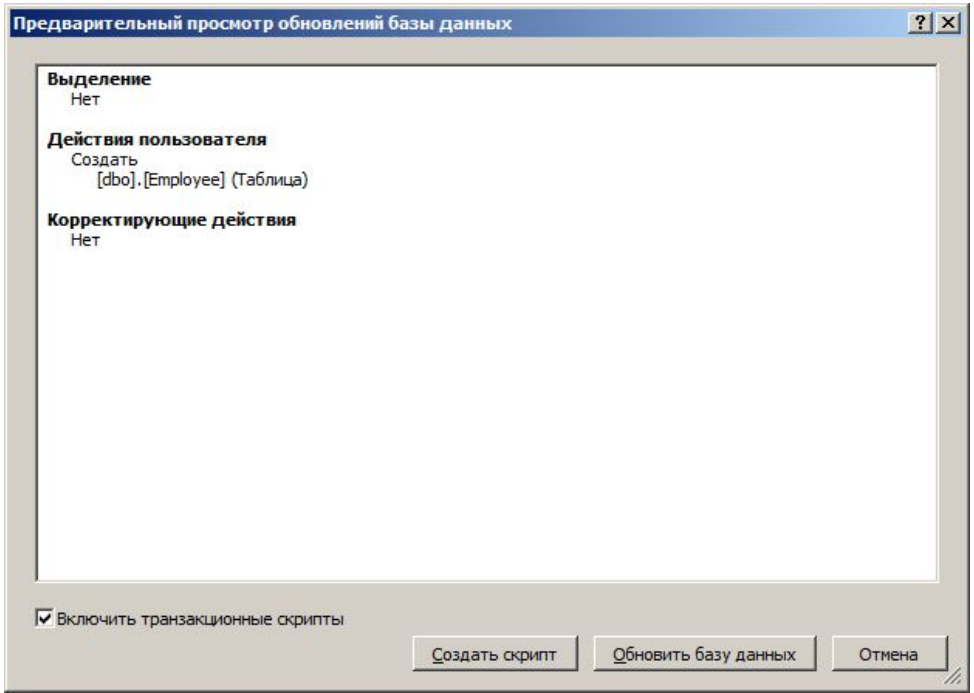
В левом верхнем углу конструктора таблиц нажмите кнопку Обновить, как показано на следующем рисунке.



20. Создание таблицы Работник (Employee)

В диалоговом окне Предварительный просмотр обновления базы данных нажмите кнопку Обновить базу данных.

Ваши изменения сохраняются в файл локальной базы данных.



21. Создание таблицы

https://msdn.microsoft.com/ru-ru/library/ms233763.aspx

11. Изъян в дизайне системы

Возможно, вы еще не забыли, что в классе PayrollDatabase были только открытые статические методы. Как теперь выясняется, это решение не годится. Как приступить к работе с реальной базой данных, не испортив все тесты, в которых использовались статические методы? Мы не хотим замещать класс PayrollDatabase классом, обращающимся к реальной базе. Если так поступить, то все ранее написанные автономные тесты станут обращаться к реальной базе данных. Хорошо бы сделать PayrollDatabase интерфейсом, чтобы можно было легко подставлять альтернативные реализации. Одна реализация хранила бы данные в памяти, чтобы выполнение тестов занимало немного времени. А другая работала бы с данными в реальной базе.

12. Рефакторинг системы

Для этого нам придется прибегнуть к рефакторингу, выполняя после каждого шага автономные тесты, чтобы случайно не испортить код.

Первым делом создадим экземпляр класса PayrollDatabase и сохраним его в статической переменной instance. Затем переименуем все статические методы в PayrollDatabase, добавив в имя слово Static. А тело каждого метода перенесем в новый не-статический метод с прежним именем (см. листинг далее).

13. Рефакторинг кода

Стр. 650!!!

Спасибо за внимание!

Расчет заработной платы: База данных – 1