МИНИСТЕРСТВО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ИРКУТСКИЙ ГОСУДАРСТВЕННЫЙ АГРАРНЫЙ УНИВЕРСИТЕТ
им. А.А. ЕЖЕВСКОГО

Кафедра «Информатика и математическое моделирование»

РАЗРАБОТКА КОРПОРАТИВНЫХ БАЗ ДАННЫХ

Учебно-методические указания
к выполнению лабораторных работ
по курсу «Технологии разработки корпоративных баз данных» для студентов направления:

09.04.03 Прикладная информатика

Молодежный, 2020

Печатается по решению методической комиссии института экономики, управления и прикладной информатики Иркутского государственного аграрного университета им. А.А. Ежевского.

Протокол №3 от 26 ноября 2020 г.

Рецензенты: к.т.н., доцент, директор института экономики, управления и прикладной информатики Федурина Н.И.; доцент кафедры информатики и математического моделирования Белякова А.Ю.

Бендик Н.В. Учебно-методические указания к выполнению лабораторных работ по курсу «Технологии разработки корпоративных баз данных» студентов направления подготовки 09.04.03 «Прикладная информатика», [Текст] / Н.В. Бендик – Иркутск: Изд-во Иркутского ГАУ, 2020. – 92 с.

Данные методические указания разработано для поддержки компьютерных лабораторных занятий и самостоятельной работы по курсу «Технологии разработки корпоративных баз данных» для студентов и магистрантов, обучающихся по направлениям «Бизнес-информатика», «Прикладная информатика».

В указаниях рассматриваются основы проектирования и построения баз данных: ER-модель и реляционная модель, а также основы языка SQL. Каждая тема содержит задания для индивидуальной работы.

В качестве среды программирования используется Microsoft SQL Server 2005 (или более поздняя версия).

Содержание

Проектирование баз данных

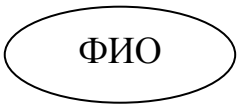
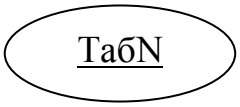
ER-модель (entity-relationship model)

Работа с базой данных начинается с построения модели предметной области. Наиболее распространенной является **ER-модель** (entity-relationship model) – модель «**Сущность-связь».**

Для построения ER-модели на практике будем использовать простую систему обозначений, предложенную Питером Ченом (обозначения, встречающиеся в разных источниках, могут несколько отличаться от нижеприведенных).

ER-модели можно просто рисовать на листочке бумаги, а также можно использовать программу **ERModeler**, разработанную одним из авторов специально для данного курса. В пособии приводятся примеры моделей, выполненные с помощью этой программы.

Базовые понятия:

Сущность (объект)	
Атрибут сущности (свойство, характеризующее объект)	
Ключевой атрибут (атрибут, входящий в первичный ключ)	
Связь	

Первичный ключ (primary key) – это атрибут или группа атрибутов, однозначно идентифицирующих объект.

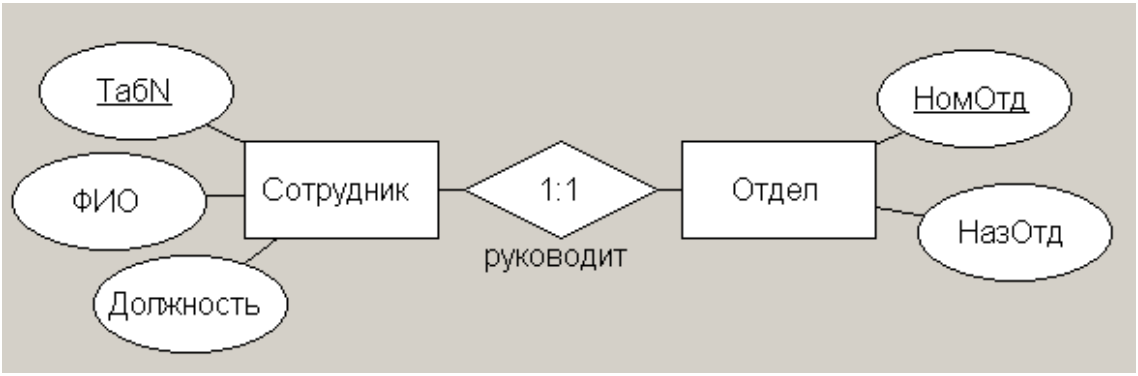
Первичный ключ может состоять из нескольких атрибутов, тогда подчеркивается каждый из них.

Объект и его атрибуты соединяются ненаправленными дугами.

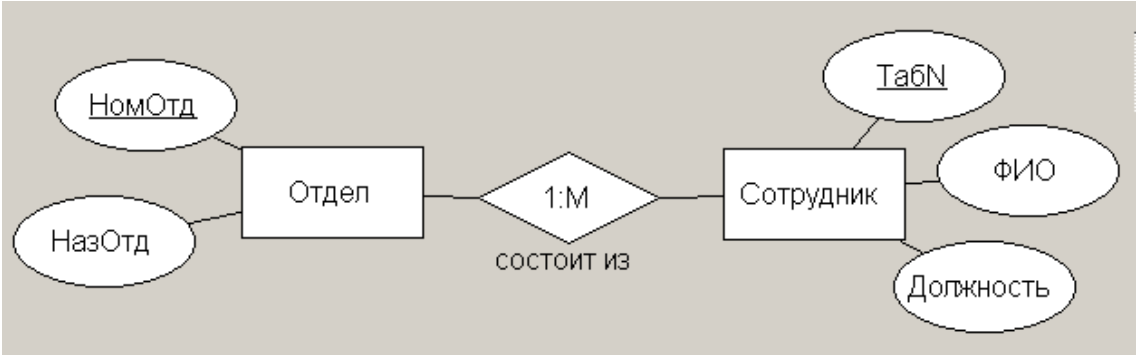
Связи между объектами могут быть 3-х типов:

Один – к одному. Этот тип связи означает, что каждому объекту первого вида соответствует не более одного объекта второго вида, и наоборот.

Например: сотрудник может руководить только одним отделом, и у каждого отдела есть только один руководитель.



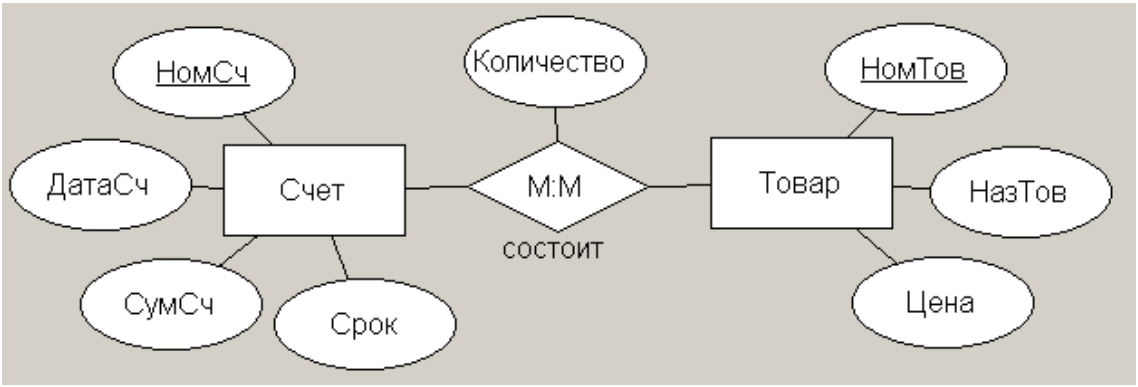
Один – ко многим *(или в обратную сторону* ***Многие – к*** *одному)*. Этот тип связи означает, что каждому объекту первого вида может соответствовать более одного объекта второго вида, но каждому объекту второго вида соответствует не более одного объекта первого вида.



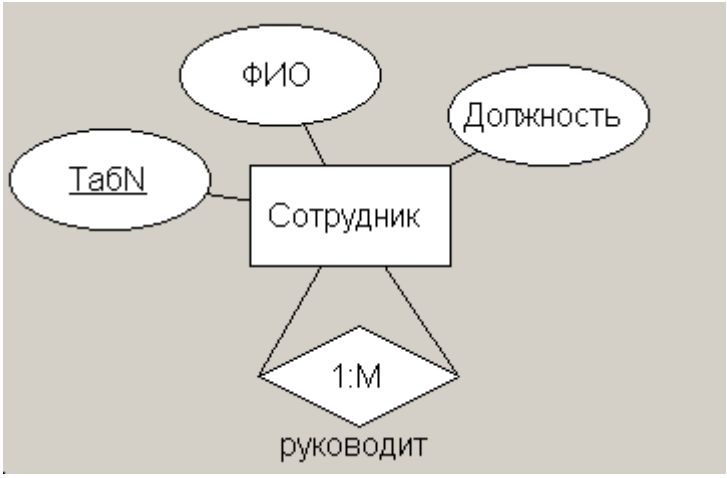
Например: в каждом отделе может быть множество сотрудников, но каждый сотрудник работает только в одном отделе.

Многие – ко многим. Этот тип связи означает, что каждому объекту первого вида может соответствовать более одного объекта второго вида, и наоборот. У этого типа связи иногда бывают собственные атрибуты.

Например: каждый счет может включать множество товаров, и каждый товар может входить в разные счета.



Связь может соединять сущность саму с собой, например:



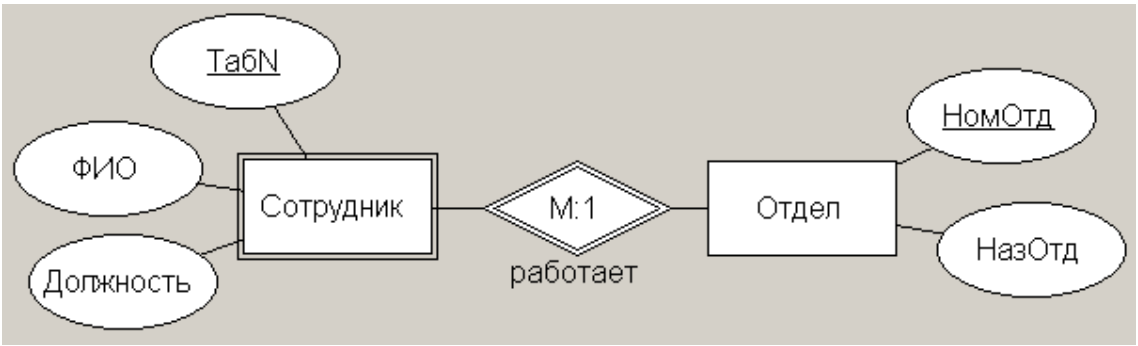
Иногда используют такое понятие, как ***слабая сущность***. Это сущность, которая не может быть однозначно идентифицирована с помощью собственных атрибутов, а только через связь с другой сущностью.

Пусть, например, номер сотрудника является уникальным только в пределах отдела, т.е. в разных отделах могут быть сотрудники с одинаковыми номерами. Уникальной в данном случае будет

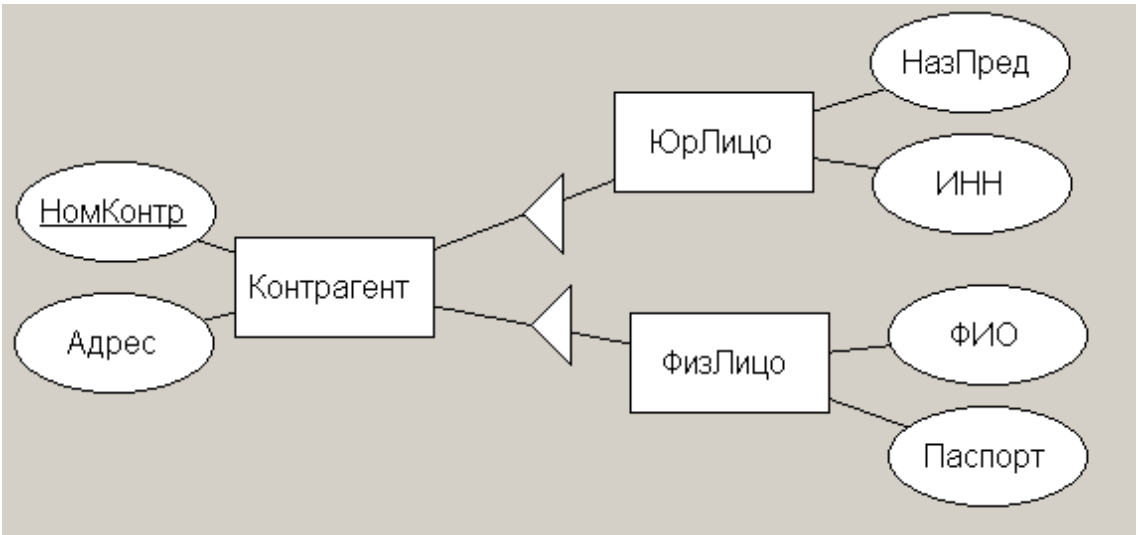
комбинация атрибутов «*НомерСотрудника, НомерОтдела*». Сущность «Сотрудник» является слабой.

На схеме слабые сущности и их идентифицирующие связи собозначаются двойными линиями.

Слабая сущность	
Связь слабой сущности	



Иногда для более удобной классификации используются так называемые ***подтипы сущностей.*** Их обозначают с помощью треугольника, направленного от подтипа к надтипу. Этот треугольник может сопровождаться надписью «есть» или «is a» (т.е., «является»).

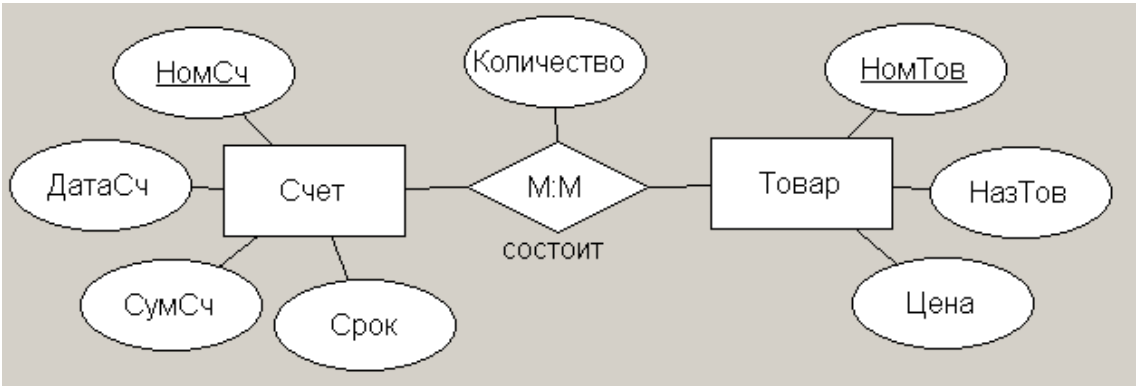


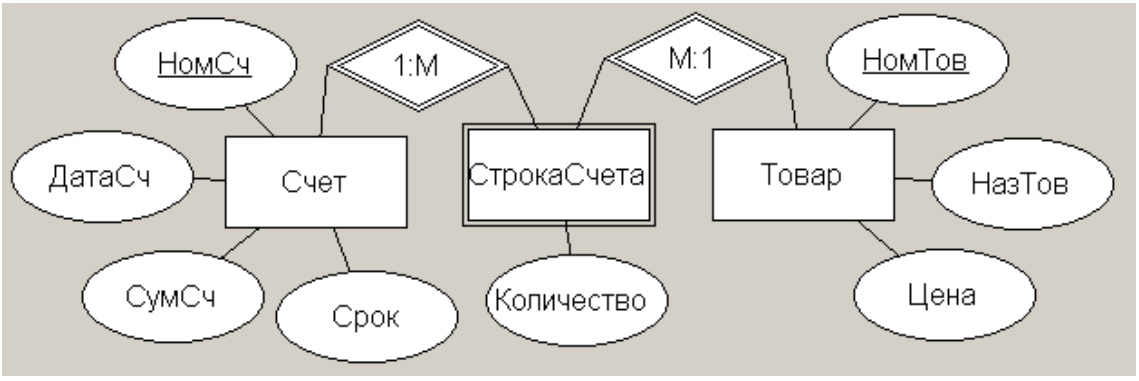
Пусть, например, среди контрагентов могут быть как физические, так и юридические лица. Поскольку они имеют разные атрибуты, то удобно создать для них подтипы.

Сущность «Контрагент» является ***надтипом*** для своих подтипов. Обратите внимание, что у подтипов обычно не бывает собственных первичных ключей.

Замечания по поводу связи M:M

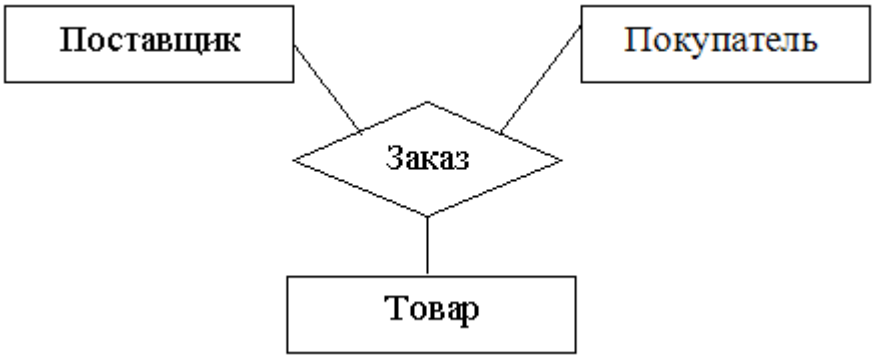
На самом деле этот тип связи представляет собой «замаскированную» слабую сущность, которая связана с другими двумя сущностями идентифицирующими связями многие – к одному:





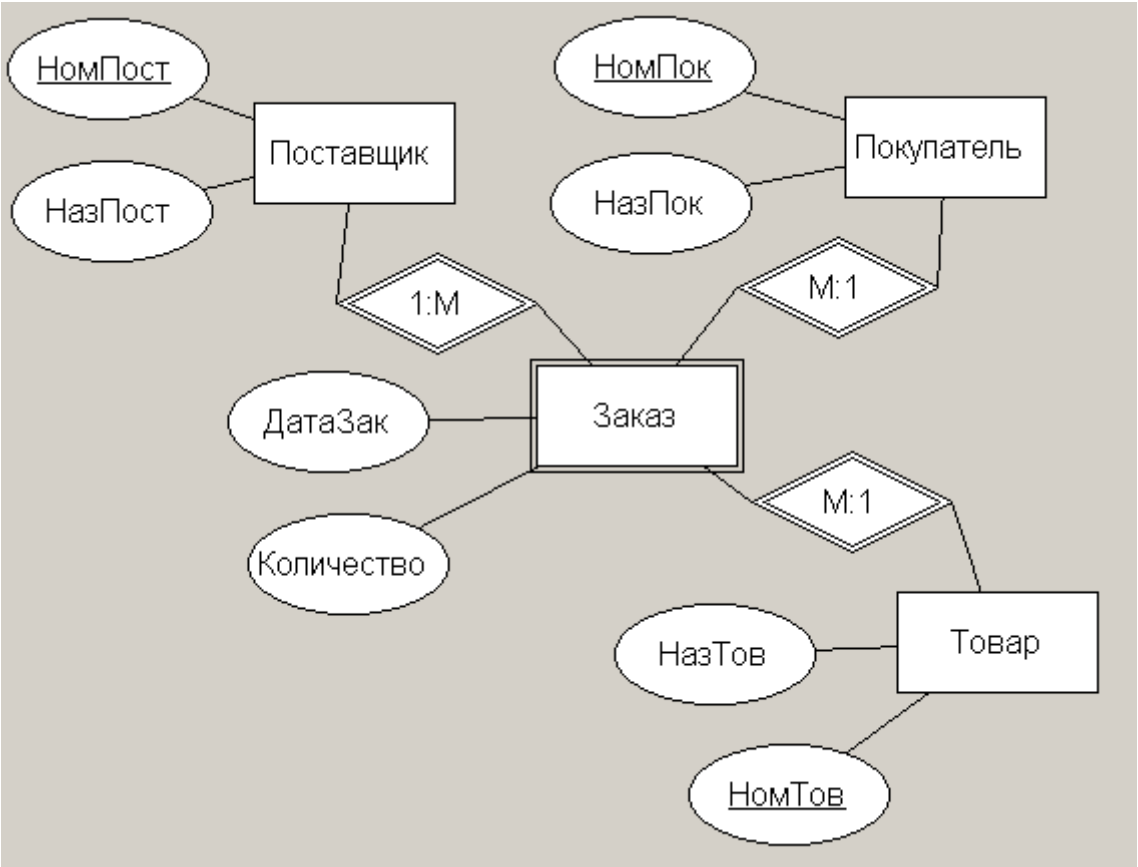
Если связь соединяет две сущности, она называется ***бинарной***.

Связь может соединять более двух сущностей, например, связь, соединяющая три сущности, называется ***тернарной:***

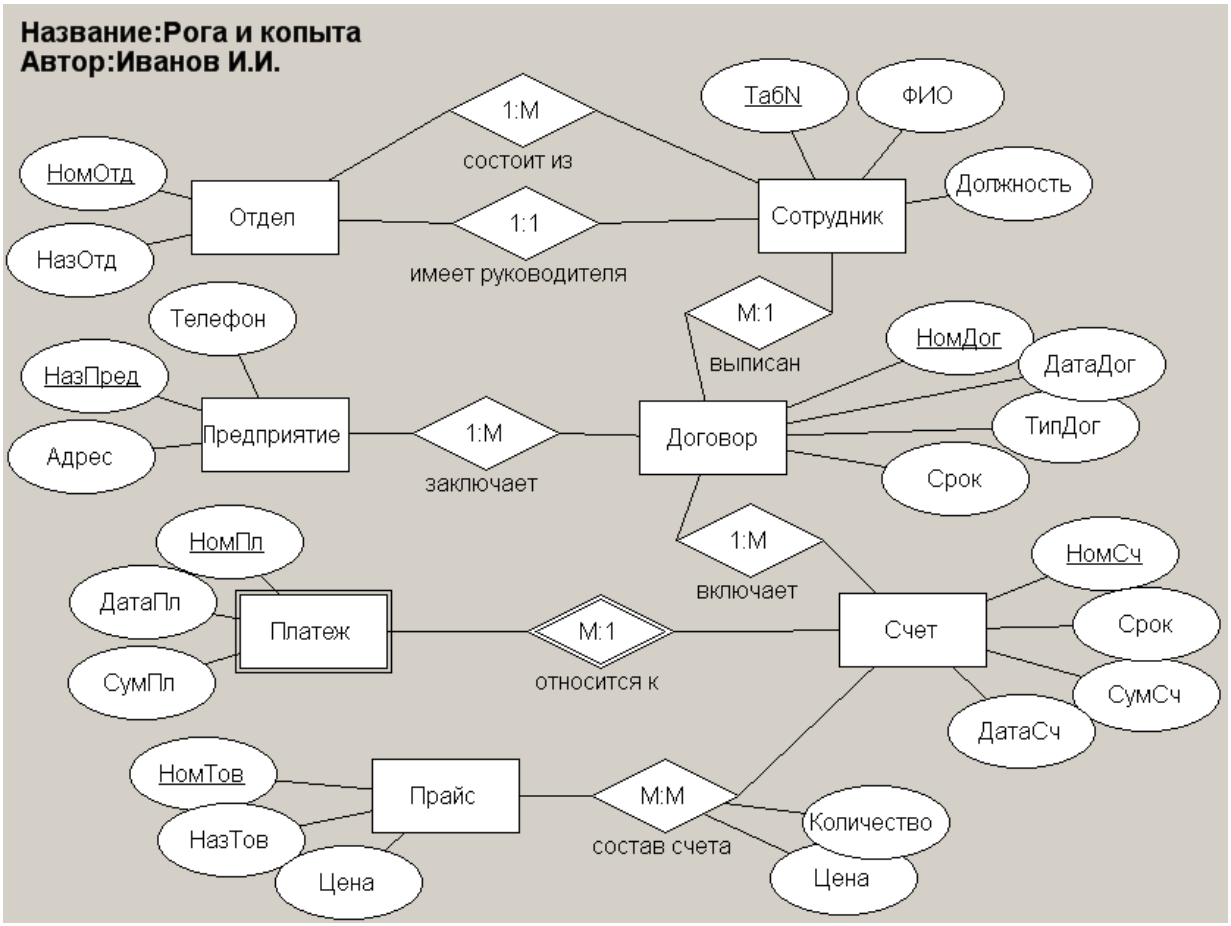


Связь с арностью более 2 обычно имеет тип **многие – ко многим** по отношению ко всем связанным сущностям.

Примечание: в программе **ERModeler** можно создавать **только бинарные** связи. Если требуется изобразить связь с большей арностью, то следует поменять её на слабую сущность:



Пример ER-модели: Контора «Рога и копыта»



Описание задачи

Контора «Рога и копыта» занимается коммерческой деятельностью по реализации продукции, произведенной из рогов и копыт, и предоставлению магических услуг.

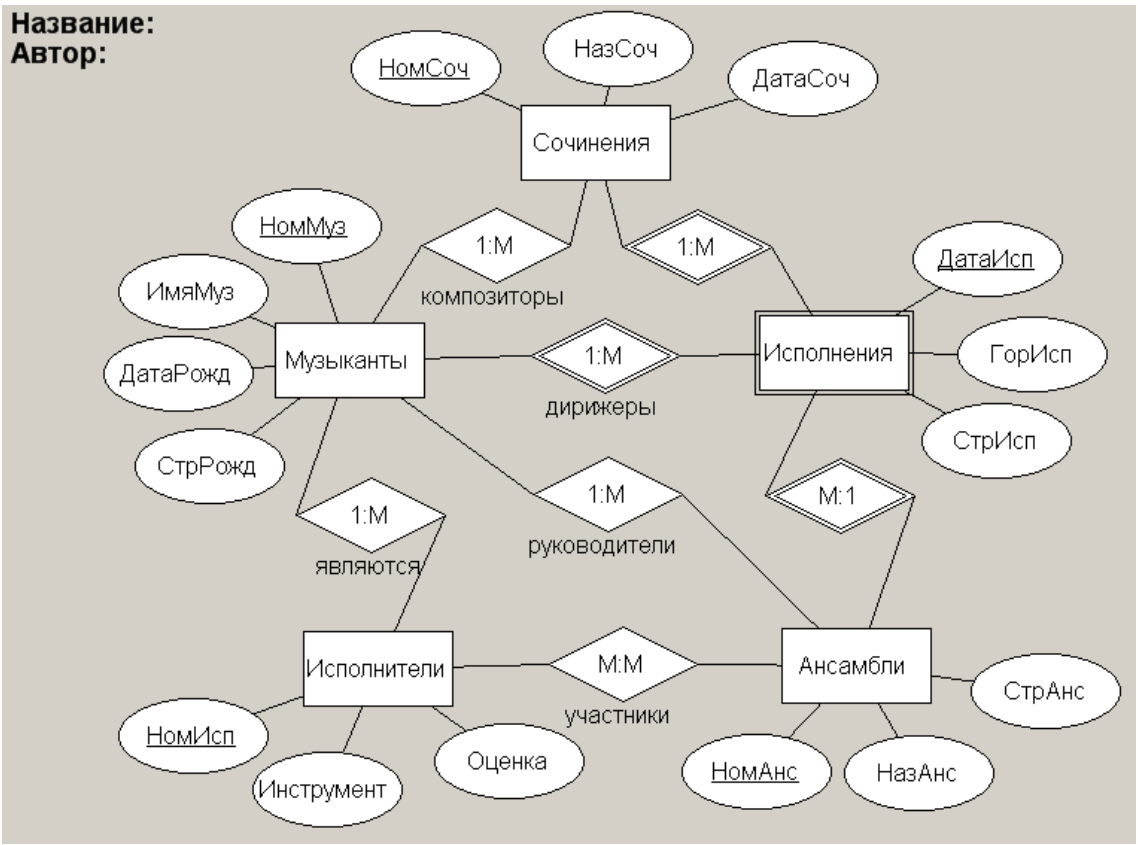
Сотрудник организации имеет ФИО, табельный номер, должность. Сотрудники распределены по нескольким отделам. Каждый отдел имеет номер, название и руководителя. Сотрудник не может руководить более чем одним отделом.

Организация работает с предприятиями-клиентами. Каждое предприятие имеет название и адрес. С предприятием может быть заключено несколько договоров.

Договор характеризуется уникальным номером, датой и типом. Каждый договор курирует некоторый сотрудник. По мере реализации клиенту товаров и услуг по договору с некоторой периодичностью выставляются счета.

Счет характеризуется уникальным номером, датой выставления, сроком оплаты и суммой, а также списком реализованных товаров и услуг с указанием их количества. По неоплаченным счетам начисляются пени. Счет может быть оплачен в несколько приемов, каждый платеж характеризуется номером, датой и суммой. Номер платежа уникален в пределах его счета. Цены на товары и услуги могут изменяться со временем.

Пример ER-модели: «Музыканты»



Описание задачи [3]

Необходимо разработать базу данных для хранения информации о музыкантах, сочинениях и концертах.

Музыкант характеризуется именем, датой рождения и страной рождения.

Сочинение включает информацию о названии, композиторе и дате первого исполнения.

Музыкант может играть на разных инструментах с разной степенью квалификации.

Из музыкантов-исполнителей формируются ансамбли. Каждый ансамбль, кроме своих участников, содержит информацию о названии, стране и руководителе.

Наконец, исполнения произведений характеризуются датой, страной, городом исполнения, а также ансамблем, дирижером и собственно исполняемым произведением.

Задание для индивидуальной работы 1

Выберите любую предметную область, для которой вы будете создавать базу данных, и разработайте для нее ER-модель. В ER-модели должно содержаться не менее 5 разных сущностей и связи между ними. Постарайтесь использовать также слабые сущности и/или подтипы сущностей.

Преобразование ER-модели в реляционную модель

Дано: ER-модель.
Получить: набор таблиц (отношений) следующего вида
Таблица (Ключ, Атрибут1, Атрибут2, …, АтрибутN)

Первичным ключом (primary key), как и в ER-модели, называется атрибут или группа атрибутов, однозначно идентифицирующих объект. Первичные ключи будем подчеркивать.

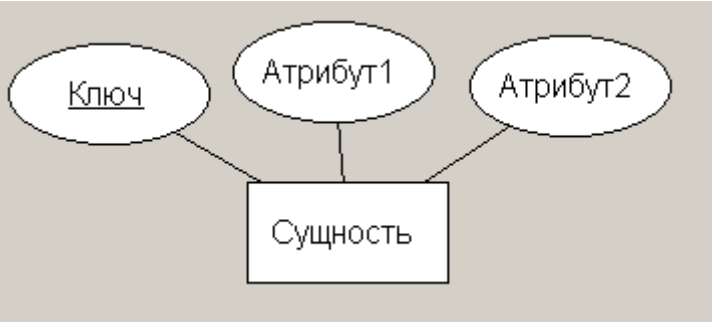
Имена атрибутов в масштабе ER-модели удобно делать уникальными, тогда при построении реляционной модели их (почти никогда) не придется переименовывать.

Внешним ключом (foreign key) называют ссылку на родительский объект. Обычно внешние ключи появляются в таблицах в результате преобразования связей. Будем выделять внешние ключи курсивом.

Для краткости в некоторых примерах пропущены несущественные неключевые атрибуты.

I. Преобразование сущностей

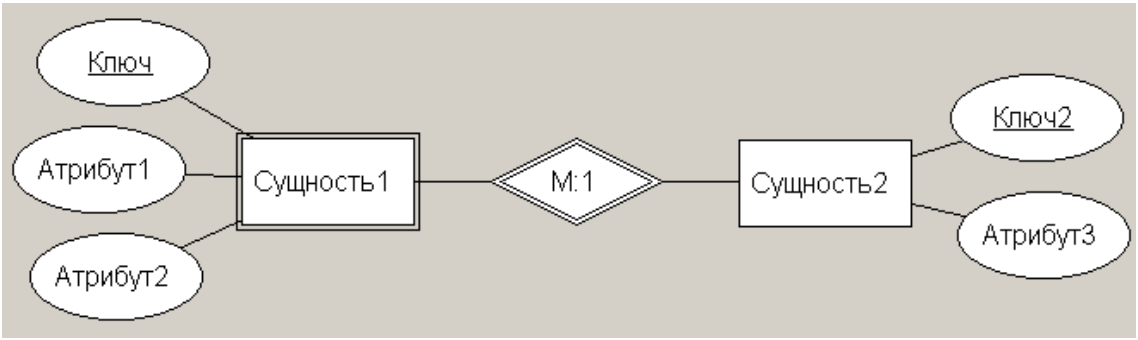
1. Преобразование обычной сущности



Обычная сущность преобразуется в отдельную таблицу, столбцами таблицы будут все атрибуты сущности:

Сущность (Ключ, Атрибут1, Атрибут2)

2. Преобразование слабой сущности



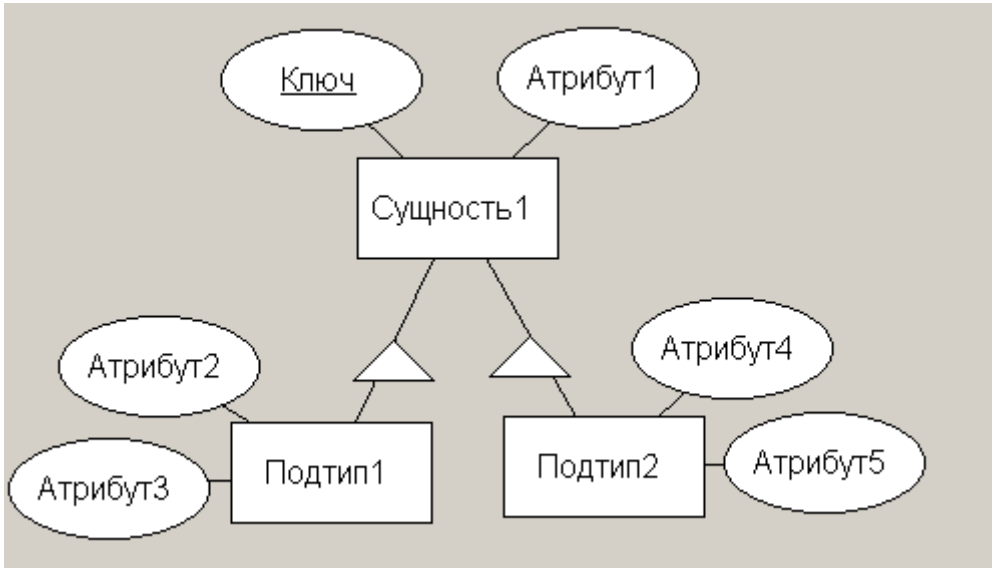
Слабая сущность преобразуется в отдельную таблицу, столбцами таблицы будут все атрибуты сущности плюс ключевые атрибуты всех сильных сущностей, с помощью которых данная слабая сущность идентифицируется.

Ключевые поля всех сильных сущностей таблиц войдут в первичный ключ слабой сущности.

Для слабой сущности они будут являться ***внешними ключами***.

Сущность1 (Ключ1, *Ключ2,* Атрибут1, Атрибут2)

3. Преобразование подтипов сущностей.



1 способ. Создается **одна** таблица, в которую помещают **все** атрибуты. Для того чтобы указать, к какому подтипу относится объект, приходится вводить дополнительное поле-признак.

Сущность1 (Ключ, Атрибут1, Атрибут2, Атрибут3, Атрибут4, Атрибут4, Признак)

Недостатком этого способа является то, что в таблице остается много незаполненных полей: для объекта подтипа 1

атрибуты 4 и 5, а для объекта подтипа 2 – атрибуты 2 и 3 останутся пустыми.

2 способ. Создается **отдельная** таблица для **каждого** подтипа. В нее включаются **все** атрибуты этого подтипа и **все** атрибуты надтипа.

Подтип1 (Ключ, Атрибут1, Атрибут2, Атрибут3)
Подтип2 (Ключ, Атрибут1, Атрибут4, Атрибут5)

Недостатком этого подхода является то, что подтипы теперь никак не связаны друг с другом.

3 способ. Создается **одна** таблица для надтипа и по одной таблице для **каждого** подтипа, в которую включаются **ключевые** поля надтипа:

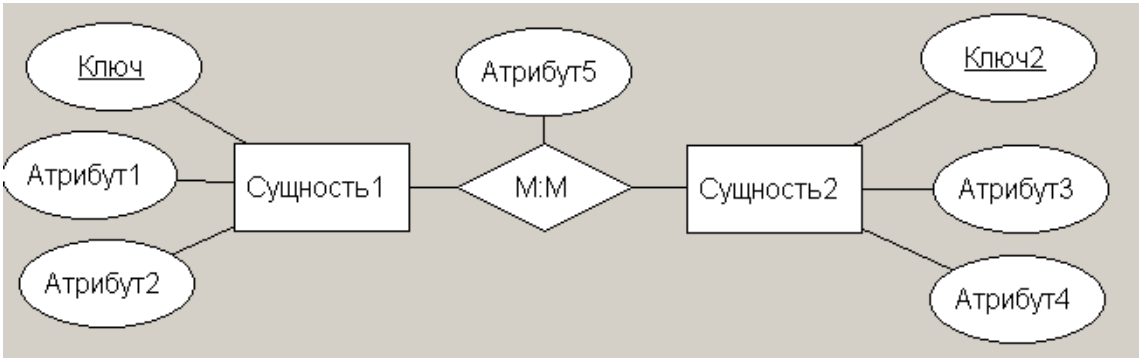
Сущность1 (Ключ, Атрибут1)
Подтип1 (*Ключ*, Атрибут2, Атрибут3)
Подтип2 (*Ключ*, Атрибут4, Атрибут5)

*Недостатком этого подхода является то, что информация о **каждом** объекте теперь распределена по двум таблицам.*

II. Преобразование связей

Для связей-двойных ромбов ничего делать не нужно, вся информация уже хранится в таблице слабой сущности.

1. Связь M:M



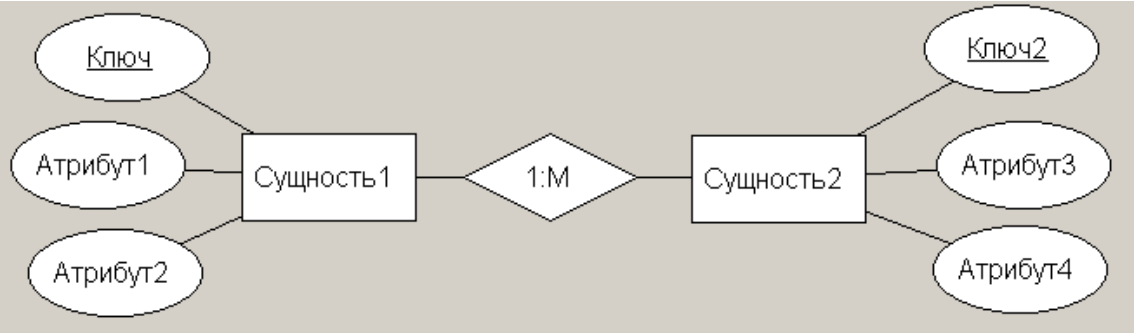
По правилам преобразования обычной сущности, как мы видели выше, для каждой сущности создается отдельная таблица, содержащая все её атрибуты:

Сущность1 (Ключ1, Атрибут1, Атрибут2)
Сущность2 (Ключ2, Атрибут3, Атрибут4)

Для связи создается **отдельная** таблица, содержащая ключевые поля каждой сущности, участвующей в связи, и собственные атрибуты связи, если таковые имеются. В названии обычно отражают, какие именно сущности связываются, или называют новую таблицу именем связи.

Сущ1Сущ2 (*Ключ1, Ключ2*, Атрибут5)

2. Связь 1:M



1 способ. Точно так же, как и в случае M:M, создаются отдельные таблицы для сущностей и отдельная таблица для связи, содержащая ключевые поля каждой сущности, участвующей в связи. Первичным ключом будет ключ второй сущности.

Сущность1 (Ключ1, Атрибут1, Атрибут2)
Сущность2 (Ключ2, Атрибут3, Атрибут4)
Сущ1Сущ2 (*Ключ1*, *Ключ2*)

Этот способ предпочтительнее использовать в том случае, если связь не является «ровно к одному», то есть не все экземпляры сущностей участвуют в связи.

2 способ. Новая таблица для связи не создается, а в таблицу дочерней сущности добавляют ключевые поля родительской сущности (в первичный ключ дочерней сущности они входить не будут!). Ключевые поля родительской сущности представляют собой ***внешний ключ*** для дочерней сущности.

Сущность1 (Ключ1, Атрибут1, Атрибут2)
Сущность2 (Ключ2, Атрибут3, Атрибут4, *Ключ1*)

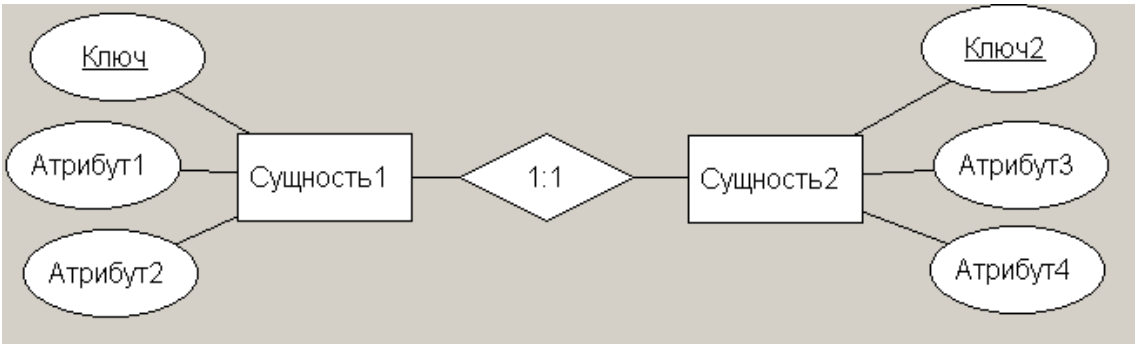
Этот способ предпочтительнее использовать в том случае, если связь является связью «ровно к одному» в сторону родительской

*сущности, то есть **все** экземпляры дочерней сущности участвуют в связи. В этом случае поле внешнего ключа никогда не будет пустым.*

3. Связь 1:1

1 способ. Точно так же, как и в случае М:М, создаются отдельные таблицы для сущностей и отдельная таблица для связи, содержащая ключевые поля каждой сущности, участвующей в связи.

Первичным ключом этой таблицы будет ключ любой сущности.



Сущность1 (Ключ1, Атрибут1, Атрибут2)
Сущность2 (Ключ2, Атрибут3, Атрибут4)
Сущ1Сущ2 (*Ключ1, Ключ2*) или **Сущ1Сущ2 (*Ключ1, Ключ2*)**

Этот способ предпочтительнее использовать в том случае, если связь не является связью «ровно к одному», то есть не все экземпляры сущностей участвуют в связи.

2 способ. Точно так же, как и во 2 случае 1:М, новая таблица для связи не создается, а в таблицу одной из сущностей (будем считать ее дочерней) добавляют ключевые поля другой сущности (будем считать ее родительской).

Сущность1 (Ключ1, Атрибут1, Атрибут2)
Сущность2 (Ключ2, Атрибут3, Атрибут4, *Ключ1*)

Если связь не является связью «ровно к одному» по отношению к родительской таблице, то есть не все экземпляры дочерней сущности участвуют в связи, поле внешнего ключа в некоторых записях может быть пустым.

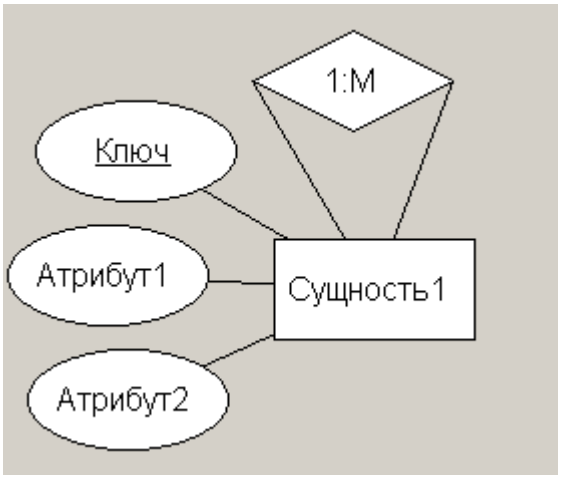
3 способ. Две таблицы для сущностей, связанных соотношением 1:1, объединяются в одну. Ключом новой таблицы может быть комбинация ключей обеих таблиц. Если хотя бы в одном направлении

связь «ровно к одному», то ключ этой сущности можно считать ключом объединенной таблицы.

Сущ1Сущ2(Ключ1, Атрибут1, Ключ2, Атрибут2)
или, возможно, **Сущ1Сущ2(Ключ1, Атрибут1, Ключ2, Атрибут2),**
или **Сущ1Сущ2(Ключ1, Атрибут1, Ключ2, Атрибут2).**

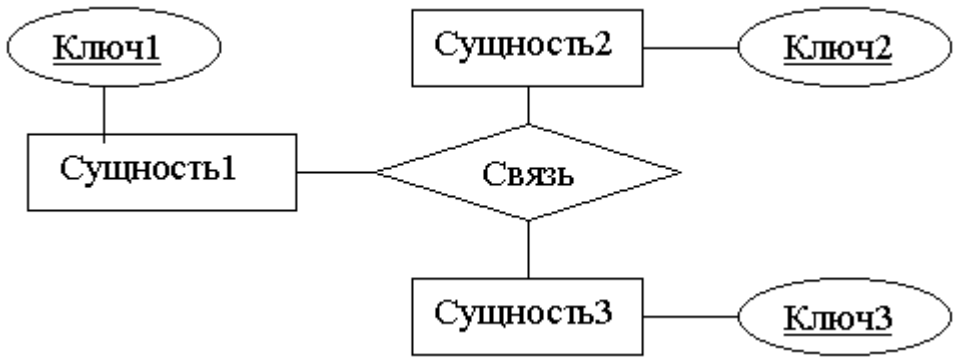
Примечание 1:

Для связи сущности с **самой собой** применяются те же правила, но так как одна и та же сущность участвует в связи дважды, ключевые поля должны войти в одну и ту же таблицу два раза. Поэтому приходится переименовывать один из ключей.



Рассмотрим связь 1:M, способ 2. Переименован внешний ключ.
Сущность1 (Ключ1, Атрибут1, Атрибут2, *ЕщеОдинКлюч1*).

Примечание 2:



Для связей с арностью более 2 обычно применяется тот же способ, что и для бинарной связи M:M – создается новая таблица, содержащая ключевые поля всех связанных таблиц.

Сущ1Сущ2Сущ3 (*Ключ1, Ключ2, Ключ3*)

Пример реляционной модели: Контора «Рога и копыта»

Таблицы удобнее называть существительными во множественном числе.

В модель добавлены дополнительные неключевые атрибуты для того, чтобы задача была более интересной.

Отделы (Ном_отд, Полное_назв_отд, Краткое_назв_отд, *Ном_сотр*)

Поле Ном_сотр содержит номер сотрудника-руководителя отдела и является результатом преобразования связи 1:1 «Имеет руководителя».

Сотрудники (Ном_сотр, ФИО, Должность, Дата_приема, Срок, *Ном_отд*)

Ном_отд появился в результате преобразования связи 1:M «Состоит из», в первичный ключ он не входит.

Предприятия (Ном_пред, Наз_пред, Адрес, Телефон)

Название предприятия неудобно использовать в качестве первичного ключа, добавим новое поле Ном_пред.

Договоры (Ном_дог, Дата_дог, Тип_дог, *Ном_сотр, Ном_пред*)

Ном_сотр появился в результате преобразования связи M:1 «Выписан», в первичный ключ он не входит, Ном_пред появился в результате преобразования связи 1:M «Заключает», в первичный ключ он не входит.

Счета (Ном_сч, Дата_сч, Срок_сч, Сумма_сч, *Ном_дог*, Пени)

Ном_дог появился в результате преобразования связи 1:M «Включает», в первичный ключ он не входит.

Платежи (Ном_пл, *Ном_сч*, Дата_пл, Сумма_пл)

*Ном_сч появился в результате преобразования слабой сущности «Платеж», **это поле входит в первичный ключ!***

Товары/услуги (Ном_ту, Назв_ту, Цена_ту)

Название товара неудобно использовать в качестве первичного ключа, добавим новое уникальное поле Ном_ту.

Протоколы_счетов (*Ном_сч, Ном_ту*, Количество, Цена)

Эта таблица появилась в результате преобразования бинарной связи М:М «Состав счета», её первичный ключ состоит из ключевых полей обеих сущностей.

Пример реляционной модели: «Музыканты»

Музыканты (НомМуз, ИмяМуз, ДатаРожд, СтрРожд)

Эта таблица не имеет внешних ключей.

Сочинения (НомСоч, НазСоч, ДатаСоч, *НомМуз*)

Столбец НомМуз является внешним ключом, появился в результате преобразования связи 1:М «Композитор» и содержит номера музыкантов-композиторов.

Исполнители (НомИсп, Инструмент, Оценка, *НомМуз*)

Столбец НомМуз является внешним ключом, появился в результате преобразования связи 1:М «Является» с сущностью «Музыканты».

Ансамбли (НомАнс, НазАнс, СтрАнс, *НомМуз*)

Столбец НомМуз является внешним ключом, появился в результате преобразования связи 1:М «Руководитель».

УчАнс(*НомАнс, НомИсп*)

Эта таблица появилась в результате преобразования бинарной связи М:М «Участники», её первичный ключ состоит из ключевых полей обеих связанных сущностей.

Исполнения (*НомМуз, НомАнс, НомСоч*, ДатаИсп, СтрИсп, ГорИсп)

Поскольку сущность «Исполнения» является слабой сущностью, в состав её первичного ключа попали все первичные ключи сильных

сущностей, от которых она зависит. По отдельности каждый из них является ещё и внешним ключом.

Задание для индивидуальной работы 2

Преобразуйте вашу ER-модель в реляционную модель. В получившихся таблицах не забудьте отметить первичные и внешние ключи.

SQL (Structured Query Language)

SQL Server – коротко о главном

В качестве среды программирования мы будем использовать СУБД (систему управления базами данных**) SQL Server** версии 2005 или выше, **Express Edition**.

SQL Server – это хорошо масштабируемый, полностью реляционный, быстродействующий многопользовательский сервер баз данных масштаба предприятия, способный обрабатывать большие объемы данных для клиент-серверных приложений. Его основные характеристики:

- многопользовательская поддержка;
- многоплатформность;
- поддержка 64-разрядной архитектуры;
- масштабируемость (многопроцессорная обработка и поддержка терабайтных БД – 10e12 байт);
- стандарт SQL92 (язык Transact SQL);
- параллельные архивирование и восстановление БД;
- репликация данных;
- распределенные запросы;
- распределенные транзакции;
- динамические блокировки;
- интеграция с IIS и Visual Studio.

Существуют дистрибутивы для разных версий операционной системы Windows, как 32-битовых, так и 64-битовых.

Установка.

1 этап. Установите **SQL Server Express** из дистрибутива. Учтите, что для установки требуется **.Net Framework 2.0** (для 2005) или более поздняя версия. Не забудьте включить текущего пользователя в группу администраторов. Для версии 2008 установка выглядит несколько сложнее; если возникают трудности, пользуйтесь справочной системой сайта Microsoft.com.

В процессе установки в общем меню программ создается папка **Microsoft SQL Server.** В этой папке в подпапке **Configuration Tools**

содержится очень полезная утилита с названием **SQL Server Configuration Manager**. С помощью этой утилиты можно запускать и останавливать сервер, а также выполнять его настройку.

2 этап. Установите **SQL Server Management Studio**. Эта среда позволяет как выполнять некоторые административные задачи с помощью визуальных средств, так и запускать SQL-сценарии в текстовом режиме.

Запустите **SQL Server Management Studio.** При запуске задается имя сервера (обычно оно выглядит как **ИмяКомпьютера\SQLEXPRESS**).

Служебные базы данных, которые создаются по умолчанию:

- **master** – системная БД с конфигурацией SQLServer;
- **model** – шаблон для всех пользовательских БД;
- **msdb** – планирование заданий по расписанию и т.п.;
- **tempdb** – БД для временных объектов.

Любая БД состоит из:

- диаграмм (автоматически не создаются!);
- таблиц (пользовательских и системных);
- представлений;
- хранимых процедур;
- пользователей;
- ролей;
- правил;
- значений по умолчанию;
- пользовательских типов.

БД можно создать:

- с помощью визуальных средств Management Studio: (Databases – New database);
- с помощью средств ER-проектирования, например Platinum ERWin,
- с помощью SQL-команд, запуская их из SQL Management Studio.

Главный файл БД (один) имеет тип .MDF. Можно задавать дополнительные файлы, их тип .NDF. Файл журнала транзакций имеет тип .LDF.

SQL Server Management Studio позволяет выполнять как отдельные SQL-команды, так и SQL-файлы целиком.

Перед выполнением запроса следует выбрать базу данных в списке в левой верхней части окна. Для выполнения файла следует нажать кнопку с восклицательным знаком или клавишу F5. Можно выполнить только часть файла, для этого ее нужно предварительно выделить.

Результаты запроса выдаются в нижнем окне. Оно имеет две закладки – для результатов и для сообщений. Результаты можно выдавать в виде текста или в виде таблицы. Если в каком-то из окон не читается русский текст, поменяйте в этом окне шрифт на русифицированный.

SQL Server Express edition не включает в себя справочную систему. Вся необходимая информация может быть найдена в онлайн-режиме в справочной системе **Microsoft MSDN** на сайте **msdn.microsoft.com.**

DDL. Таблицы

DDL (Data Definition Language) – язык описания данных, составная часть SQL. Рассмотрим команды создания базы данных и таблиц.

Для создания базы данных служит команда

```
CREATE DATABASE имя_БД
```

Для активизации базы данных служит команда

```
USE имя_БД
```

Выполняйте команду активизации базы данных при каждом запуске SQL Management Studio, поскольку по умолчанию в качестве активной установлена БД `master`.

Для создания таблиц используется команда `CREATE TABLE`.

Краткий формат этой команды (квадратные скобки означают необязательные элементы):

```
CREATE TABLE имя_таблицы(
              Список_определений_полей,
              [Список_ограничений_таблицы]);
```

Более подробно смотрите в **MSDN**.

Определение поля имеет формат:

```
Имя_поля тип_поля[(размер)]
              [NULL] [NOT NULL]
              [IDENTITY]
              [DEFAULT умолчание]
              [Список_ограничений_поля]
```

Ограничение поля имеет формат

```
[CHECK (условие)]
[PRIMARY KEY]
[UNIQUE]
[REFERENCES имя_таблицы(имя_поля)]
```

Чаще всего используются типы полей:

`VARCHAR` – строковый тип переменной длины;

`NUMERIC` – числовой тип;

`DATETIME` – тип дата/время.

Какие еще типы полей есть в SQL server? – обращайтесь ***к MSDN.***

NULL – специальное «неопределенное» значение, предусмотренное стандартом SQL. Определение NULL/NOT NULL служит для указания, что данный тип поля допускает/запрещает ввод NULL-значений.

IDENTITY начальное_значение, приращение – определение, указывающее, что данное поле представляет собой счетчик. Это означает, что значения в данное поле вставляются сервером с нарастанием автоматически при вставке строки. Если «начальное_значение» и «приращение» пропущены, они полагаются равными 1.

DEFAULT умолчание – определение, указывающее значение по умолчанию, т.е., значение, которое присваивается данному полю, если при вставке новой строки этому полю не было явно присвоено некоторое значение.

CHECK (условие)– ограничение, содержащее условие на поле, которое будет проверяться при вводе новых строк и при изменении значений в уже существующих строках. Если при добавлении или изменении строки условие оказывается ложным, то строка не добавляется/ не изменяется.

PRIMARY KEY – ограничение, указывающее, что в данной таблице данное поле представляет собой первичный ключ. (Составной первичный ключ таким образом объявлять нельзя!) При использовании этого ограничения создается *первичный индекс*.

UNIQUE – ограничение, указывающее, что в данном поле могут храниться только уникальные значения. При использовании этого ограничения создается *уникальный индекс*.

Например, в таблице «Предприятия» номер предприятия будет первичным ключом и счетчиком, название фирмы не допускает значений NULL:

```
CREATE TABLE k_firm
    (firm_num  NUMERIC(6) IDENTITY PRIMARY KEY,
    firm_name VARCHAR(100)    NOT NULL,
    firm_addr  VARCHAR(100)
);
```

В таблице «Договоры» для поля даты договора задается значение по умолчанию – текущая дата, для типа договора задается условие, что он должен принадлежать заданному списку значений.

```
CREATE TABLE k_contract
    contract_num    NUMERIC(6) IDENTITY PRIMARY KEY,
    contract_date   DATETIME  DEFAULT GETDATE(),
    contract_type   CHAR(1)
         CHECK (contract_type IN ('A','B','C')),
    firm_num        NUMERIC(6) NOT NULL,
    staff_num       NUMERIC(6)
);
```

`REFERENCES имя_таблицы(имя_поля)` – ограничение *внешнего ключа*, или *декларативной ссылочной целостности.*

Декларативная ссылочная целостность требует, чтобы в поле *внешнего ключа* можно было вводить только такие значения *первичного ключа*, которые присутствуют в родительской таблице. *Например*, в таблицу «Сотрудники» мы не можем внести номер несуществующего отдела. Кроме того, из родительской таблицы нельзя удалить строку, если в дочерней таблице имеются строки с таким *внешним ключом*. Мы не можем удалить отдел, если с ним связаны сотрудники.

```
CREATE TABLE k_staff
    (staff_num NUMERIC(6) IDENTITY,
    staff_name VARCHAR(30) NOT NULL,
    staff_post VARCHAR(30),
    dept_num        NUMERIC(6)
         REFERENCES k_dept (dept_num),
    staff_hiredate DATETIME NOT NULL,
    staff_termdate DATETIME
);
```

Ограничения уровня таблицы задаются после списка определений полей. Каждое из них содержит ключевое слово `CONSTRAINT` и уникальное имя. Эти ограничения применяются обычно в том случае, если включают в себя несколько полей, например, составной первичный или внешний ключ или условие `CHECK`, содержащее сразу несколько полей таблицы. *Следует заметить, что любое ограничение уровня поля можно переписать как ограничение уровня таблицы.*

Ограничение `CHECK` уровня таблицы может быть определено, например, так:

```
CREATE TABLE k_bill
    (bill_num       NUMERIC(6) IDENTITY PRIMARY KEY,
    bill_date       DATETIME  DEFAULT GETDATE(),
    bill_term       DATETIME  DEFAULT GETDATE()+30,
    contract_num    NUMERIC(6),
    CONSTRAINT ch_bill_date  CHECK (bill_term-bill_date<91)
);
```

т.е., срок действия счета не может превышать 91 день.

Ограничение *внешнего ключа* на уровне таблицы определяется так:

```
    CONSTRAINT имя_ограничения FOREIGN KEY (список_полей)
         REFERENCES родительская_таблица(внешний ключ)
```

например,

```
CREATE TABLE k_contract
    (contract_num   NUMERIC(6) IDENTITY PRIMARY KEY,
    contract_date   DATETIME  DEFAULT GETDATE(),
    contract_type   CHAR(1)
         CHECK (contract_type IN ('A','B','C')),
    firm_num   NUMERIC(6) NOT NULL,
    staff_num  NUMERIC(6),
    CONSTRAINT fk_contract_firm_num FOREIGN KEY (firm_num)
               REFERENCES k_firm (firm_num),
    CONSTRAINT fk_contract_staff_num FOREIGN KEY (staff_num)
     REFERENCES k_staff (staff_num)
              );
```

т.е, для таблицы договоров есть два *различных* внешних ключа: номер предприятия и номер сотрудника, ссылающиеся на таблицы «Предприятия» и «Сотрудники».

В том случае, когда первичный ключ состоит из нескольких полей, его придется создавать как ограничение уровня таблицы:

```
    CONSTRAINT имя_ограничения PRIMARY KEY (список_полей)
```

Например, в таблице протоколов счета первичный ключ состоит из двух полей, на основе каждого из них также создается внешний ключ:

```
CREATE TABLE k_protokol
     price_num NUMERIC(6) NOT NULL ,
     bill_num   NUMERIC(6) NOT NULL ,
     kolvo      NUMERIC(6) NOT NULL ,
```

```
     price_sum  NUMERIC(9,2) NOT NULL ,
    CONSTRAINT pk_protokol_num
         PRIMARY KEY (price_num, bill_num),
    CONSTRAINT fk_protokol_price_num FOREIGN KEY (price_num)
     REFERENCES k_price (price_num),
    CONSTRAINT fk_protokol_bill_num FOREIGN KEY (bill_num)
     REFERENCES k_bill (bill_num)
);
```

Кроме команды `CREATE TABLE`, к секции DDL относятся также команды `ALTER TABLE` (изменение описания таблицы) и `DROP TABLE` (удаление таблицы). Так, например, с помощью команды `ALTER TABLE` можно добавлять или удалять столбцы или ограничения для уже созданной таблицы. Подробнее об этих командах можно прочитать в **MSDN**, а примеры их использования приведены в следующем параграфе.

Пример сценария создания БД "РОГА И КОПЫТА"

Рассмотрим полностью сценарий создания базы данных для фирмы «Рога и копыта». Сначала создаются родительские таблицы, затем дочерние, т.е., такие, которые содержат ограничения внешних ключей.

```
CREATE DATABASE kontora
    USE kontora
```

Таблица "Предприятия"

```
CREATE TABLE k_firm
       (firm_num     NUMERIC(6) IDENTITY PRIMARY KEY,
       firm_name     VARCHAR(100)  NOT NULL,
       firm_addr     VARCHAR(100),
       firm_phone    NUMERIC(7)
    )
```

Таблица "Отделы"

Мы не можем пока определить внешний ключ для поля staff_num, так как таблица "Сотрудники" еще не определена.

```
CREATE TABLE k_dept
       (dept_num     NUMERIC(6) IDENTITY PRIMARY KEY,
       dept_short_name      VARCHAR(10) NOT NULL,
       dept_full_name       VARCHAR(100),
       staff_num            NUMERIC(6)
    )
```

Таблица "Сотрудники"

После создания этой таблицы сразу же можем определить внешний ключ для поля staff_num таблицы k_dept. Это можно сделать с помощью команды `ALTER TABLE`.

```
CREATE TABLE k_staff
       (staff_num    NUMERIC(6)    IDENTITY,
       staff_name    VARCHAR(30)   NOT NULL,
       staff_post    VARCHAR(30),
```

```
      dept_num     NUMERIC(6) ,
      staff_hiredate DATETIME    NOT NULL,
      staff_termdate DATETIME,
   CONSTRAINT pk_staff_num PRIMARY KEY (staff_num),
   CONSTRAINT fk_staff_dept_num FOREIGN KEY (dept_num)
      REFERENCES k_dept (dept_num)
   )

ALTER TABLE k_dept ADD CONSTRAINT fk_staff_num
      FOREIGN KEY (staff_num)
      REFERENCES k_staff(staff_num)
```

Таблица "Договоры"

```
CREATE TABLE k_contract
      (contract_num NUMERIC(6) IDENTITY PRIMARY KEY,
      contract_date DATETIME  DEFAULT GETDATE(),
      contract_type CHAR(1)
            CHECK (contract_type IN ('A','B','C')),
      firm_num      NUMERIC(6) NOT NULL,
      staff_num     NUMERIC(6),
   CONSTRAINT fk_contract_firm_num FOREIGN KEY (firm_num)
                     REFERENCES k_firm (firm_num),
   CONSTRAINT fk_contract_staff_num FOREIGN KEY (staff_num)
                  REFERENCES k_staff (staff_num)
   )
```

Таблица "Счета"

ALTER TABLE *здесь приводится просто для иллюстрации, как можно добавлять столбцы в уже созданную таблицу.*

```
CREATE TABLE k_bill
      (bill_num    NUMERIC(6) IDENTITY PRIMARY KEY,
      bill_date    DATETIME  DEFAULT GETDATE(),
      bill_term    DATETIME  DEFAULT GETDATE()+30,
      bill_peni    NUMERIC(6) DEFAULT 0,
      contract_num NUMERIC(6),
   CONSTRAINT fk_bill_contract_num
      FOREIGN KEY (contract_num)
      REFERENCES k_contract (contract_num),
   CONSTRAINT ch_bill_date  CHECK (bill_term-bill_date<91)
   )

ALTER TABLE k_bill ADD bill_sum NUMERIC(6) DEFAULT 0 NOT NULL
```

Таблица "Платежи"

Первичный ключ здесь состоит из нескольких полей, поэтому ограничение `PRIMARY KEY` *можно создавать только на уровне таблицы, а не на уровне поля.*

```
CREATE TABLE k_payment
       (payment_num NUMERIC(2) DEFAULT 0,
       bill_num     NUMERIC(6),
       payment_date DATETIME  DEFAULT GETDATE(),
       payment_sum  NUMERIC(9,2),
    CONSTRAINT pk_payment_num
              PRIMARY KEY (payment_num, bill_num),
    CONSTRAINT fk_payment_bill_num FOREIGN KEY (bill_num)
                    REFERENCES k_bill (bill_num)
    )
```

Таблица "Товары/услуги" (или "Прайс-лист")

```
CREATE TABLE k_price
       (price_num   NUMERIC(6) IDENTITY PRIMARY KEY,
       price_name   VARCHAR(100) NOT NULL,
       price_sum    NUMERIC(9,2),
       type_num     NUMERIC(6)
    )
```

Таблица "Протоколы счетов"

Первичный ключ здесь также состоит из нескольких полей, поэтому ограничение PRIMARY KEY можно создавать только на уровне таблицы, а не на уровне поля. Для каждого из этих полей здесь также создается ограничение внешнего ключа.

```
CREATE TABLE k_protokol
       (price_num   NUMERIC(6)   NOT NULL ,
       bill_num     NUMERIC(6)   NOT NULL ,
       kolvo        NUMERIC(6)   NOT NULL ,
       price_sum    NUMERIC(9,2),
    CONSTRAINT pk_protokol_num
       PRIMARY KEY (price_num, bill_num),
    CONSTRAINT fk_protokol_price_num FOREIGN KEY (price_num)
       REFERENCES k_price (price_num),
    CONSTRAINT fk_protokol_bill_num FOREIGN KEY (bill_num)
```

```
      REFERENCES k_bill (bill_num)
    )
```

В процессе отладки сценария создания базы данных вам наверняка не раз придется удалять таблицы и создавать их заново. Поэтому для удаления таблиц удобно написать отдельный сценарий.

Перед удалением каждой таблицы выполняется проверка – существует ли эта таблица. Информацию обо всех объектах БД можно получить из системной таблицы **sysobjects**. Тип объекта базы данных ‘U’ означает ‘user table’, т.е., пользовательская таблица, ‘F’ – ‘foreigh key’, т.е., внешний ключ. Для нашей базы данных сценарий может выглядеть следующим образом:

```
IF EXISTS( SELECT name FROM sysobjects
    WHERE name='k_payment' AND type='U')
     DROP TABLE k_payment
IF EXISTS( SELECT name FROM sysobjects
    WHERE name='k_protokol' AND type='U')
     DROP TABLE k_protokol
IF EXISTS( SELECT name FROM sysobjects
    WHERE name='k_bill' AND type='U')
     DROP TABLE k_bill
IF EXISTS( SELECT name FROM sysobjects
    WHERE name='k_price' AND type='U')
     DROP TABLE k_price
IF EXISTS( SELECT name FROM sysobjects
    WHERE name='k_contract' AND type='U')
     DROP TABLE k_contract
IF EXISTS( SELECT name FROM sysobjects
    WHERE name='fk_staff_num' AND type='F')
     ALTER TABLE k_dept DROP CONSTRAINT fk_staff_num
IF EXISTS( SELECT name FROM sysobjects
    WHERE name='k_staff' AND type='U')
     DROP TABLE k_staff
IF EXISTS( SELECT name FROM sysobjects
    WHERE name='k_dept' AND type='U')
     DROP TABLE k_dept
IF EXISTS( SELECT name FROM sysobjects
    WHERE name='k_firm' AND type='U')
     DROP TABLE k_firm
IF EXISTS( SELECT name FROM sysobjects
    WHERE name='k_bill_list' AND type='U')
     DROP TABLE k_bill_list
```

Удаляются сначала дочерние таблицы, затем родительские.

Таблицы "Отделы" и "Сотрудники" взаимно ссылаются друг на друга по внешним ключам, поэтому сначала приходится удалить одно из ограничений внешнего ключа, и только потом удалять таблицы. *Можно ли в данном случае поступить наоборот, т.е. удалить ограничение из таблицы k_staff? Что еще нужно будет изменить в сценарии?*

Вопрос.

Чем отличаются ограничения уровня поля и ограничения уровня таблицы?

Задание для индивидуальной работы 3

Напишите и отладьте SQL-сценарий создания вашей базы данных и таблиц для нее.

DML. Изменение данных

DML (Data Manipulation Language) – язык манипулирования данными, составная часть SQL.

Рассмотрим его основные команды – команды добавления, изменения и удаления данных `INSERT, UPDATE` и `DELETE`.

Для добавления новых строк в таблицу служит команда `INSERT`:

```
INSERT [INTO] имя_таблицы [(список_полей)]
    VALUES (список_значений);
```

Например,

```
INSERT k_firm (firm_name, firm_addr)
    VALUES('Альфа', 'Москва');
```

Список полей можно явно не указывать, тогда в списке значений нужно задавать значения для каждого поля в том порядке, в котором они были созданы.

Для поля с ограничением `IDENTITY` обычно явное значение указывать нельзя, т.к. оно формируется автоматически. Если требуется явно указывать значения для таких полей, следует предварительно выполнить команду:

```
SET IDENTITY_INSERT ON
```

Если мы вставляем значения из одной таблицы в другую, формат команды `INSERT` следующий:

```
INSERT [INTO] имя_таблицы [(список_полей)]
    (SELECT параметры);
```

Для обновления данных используется команда `UPDATE`:

```
UPDATE имя_таблицы
    SET поле1=выражение1 [,…, полеN=ВыражениеN]
    [WHERE условие];
```

Например,

```
UPDATE k_dept SET staff_num=1
    WHERE dept_short_name='Sales';
```

Если опция `WHERE` пропущена, изменяться будут *все* строки таблицы.

Для удаления данных используется команда `DELETE`:

```
DELETE [FROM] имя_таблицы [WHERE условие];
```

Например,

```
DELETE FROM k_dept WHERE dept_short_name='Sales';
```

Если опция `WHERE` пропущена, удалены будут *все* строки таблицы.

Заполним тестовыми данными нашу базу «Рога и копыта». Кроме команд добавления, для примера рассмотрены несколько команд изменения данных. Обратите внимание, что для полей, имеющих свойство `IDENTITY`, значения не задаются – они будут генерироваться автоматически.

Строки-константы следует задавать в одинарных кавычках.

Могут возникать некоторые проблемы с заданием констант типа дата. Формат таких констант зависит от региональных настроек операционной системы.

Если вы хотите задать определенный формат даты, например, **день:месяц:год**, выполните команду:

```
SET DATEFORMAT dmy
```

Мы будем использовать формат **год:месяц:день**

```
SET DATEFORMAT ymd
```

при котором константа-дата выглядит так: ‘2012-01-31’

В примерах также используется функция `GETDATE()`, которая возвращает текущие дату/время.

Таблица "Предприятия"

```
INSERT INTO k_firm (firm_name, firm_addr)
    VALUES('Альфа', 'Москва');
INSERT INTO k_firm (firm_name, firm_addr)
    VALUES('Бета', 'Казань');
INSERT INTO k_firm (firm_name, firm_addr)
    VALUES('Гамма', 'Париж');
INSERT INTO k_firm (firm_name, firm_addr)
         VALUES('Дельта', 'Лондон');
INSERT INTO k_firm (firm_name, firm_addr)
    VALUES('Омега', 'Токио');
```

Посмотрим результат заполнения, для этого выполним следующую команду:

```
SELECT * FROM k_firm;
```

Результат будет выглядеть следующим образом (поле `firm_phone` мы не заполняли, поэтому в нем будут значения `NULL`):

```
firm_num firm_name  firm_addr  firm_phone
-----------------------------------------
1        Альфа      Москва     NULL
2        Бета       Казань     NULL
3        Гамма      Париж      NULL
4        Дельта     Лондон     NULL
5        Омега      Токио      NULL

(5 row(s) affected)
```

Таблица "Отделы"

```
INSERT INTO k_dept (dept_short_name, dept_full_name)
    VALUES('Sales', 'Отдел продаж');
INSERT INTO k_dept (dept_short_name, dept_full_name)
    VALUES('Mart', 'Отдел маркетинга');
INSERT INTO k_dept (dept_short_name, dept_full_name)
     VALUES('Cust', 'Отдел гарантийного обслуживания');
```

Посмотрим результат заполнения, для этого выполним следующую команду:

```
    SELECT * FROM k_dept;
```

```
dept_num dept_short_name dept_full_name    staff_num
-----------------------------------------------------
1     Sales  Отдел продаж                    NULL
2     Mart   Отдел маркетинга                NULL
3     Cust   Отдел гарантийного обслуживания NULL

(3 row(s) affected)
```

Таблица "Сотрудники"

```
INSERT INTO k_staff
     (staff_name, dept_num, staff_hiredate, staff_post)
     VALUES('Иванов', 1, '1999-01-01', 'Менеджер');
INSERT INTO k_staff
     (staff_name, dept_num, staff_hiredate, staff_post)
     VALUES('Петров', 2, '2010-10-13','Менеджер');
INSERT INTO k_staff
     (staff_name, dept_num, staff_hiredate, staff_post)
     VALUES('Сидоров', 3, '2005-12-01','Менеджер');
INSERT INTO k_staff
     (staff_name, staff_hiredate, staff_post)
     VALUES('Семенов', '1990-01-01','Директор');
```

```
INSERT INTO k_staff
     (staff_name, dept_num, staff_hiredate, staff_post)
     VALUES('Григорьев', 3, '2008-12-19','Программист');
```

Посмотрим результат заполнения, для этого выполним следующую команду:

```
SELECT * FROM k_staff;
```

```
 staff  staff   staff  dept  staff                staff
_num   _name   _post  _num  _hiredate           _termdate
------------------------------------ -------------------
1   Иванов    Менеджер    1   1999-01-01 00:00:00.000  NULL
2   Петров    Менеджер    2   2010-10-13 00:00:00.000  NULL
3   Сидоров   Менеджер    3   2005-12-01 00:00:00.000  NULL
4   Семенов   Директор   NULL 1990-01-01 00:00:00.000  NULL
5   Григорьев Программист 3   2008-12-19 00:00:00.000  NULL

(3 row(s) affected)
```

После того как мы заполнили таблицу "Сотрудники", мы можем в таблице "Отделы" заполнить столбец `staff_num`, содержащий код руководителя отдела.

```
UPDATE k_dept SET staff_num=2
    WHERE dept_short_name='Mart';
UPDATE k_dept SET staff_num=3
    WHERE dept_short_name='Cust';
UPDATE k_dept SET staff_num=1
    WHERE dept_short_name='Sales';
```

Посмотрим результат изменения, для этого выполним следующую команду:

```
SELECT * FROM k_dept;
```

```
dept_num dept_short_name   dept_full_name     staff_num
-----------------------------------------------------
1        Sales    Отдел продаж                    1
2        Mart     Отдел маркетинга                2
3        Cust     Отдел гарантийного обслуживания 3

(3 row(s) affected)
```

Таблица "Договоры"

```
INSERT INTO k_contract
    (contract_type, firm_num, staff_num, contract_date)
    VALUES('A', 1, 1,'2011-11-01');
INSERT INTO k_contract
    (contract_type, firm_num, staff_num, contract_date)
    VALUES('B', 1, 2,'2011-10-01');
INSERT INTO k_contract
    (contract_type, firm_num, staff_num, contract_date)
    VALUES('C', 1, 1,'2011-09-01');
INSERT INTO k_contract
    (contract_type, firm_num, staff_num, contract_date)
    VALUES('A', 2, 2,'2011-11-15');
INSERT INTO k_contract
    (contract_type, firm_num, staff_num, contract_date)
    VALUES('B', 2, 2,'2011-08-01');
INSERT INTO k_contract
    (contract_type, firm_num, staff_num, contract_date)
    VALUES('C', 3, 1,'2011-07-15');
INSERT INTO k_contract
    (contract_type, firm_num, staff_num, contract_date)
    VALUES('A', 4, 1,'2011-11-12');
```

Посмотрим результат заполнения, для этого выполним следующую команду:

```
SELECT * FROM k_contract;
```

```
contract_num contract_date contract_type firm_num staff_num
-------------------------------------------------------------
1         2011-11-01 00:00:00.000   A     1        1
2         2011-10-01 00:00:00.000   B     1        2
3         2011-09-01 00:00:00.000   C     1        1
4         2011-11-15 00:00:00.000   A     2        2
5         2011-08-01 00:00:00.000   B     2        2
6         2011-07-15 00:00:00.000   C     3        1
7         2011-11-12 00:00:00.000   A     4        1

(7 row(s) affected)
```

Обратите внимание, что даты договоров заполнились автоматически текущими датой и временем – это сработало определение `DEFAULT` для данного поля.

Таблица "Счета"

```
INSERT INTO k_bill
```

```
    (contract_num, bill_date, bill_term, bill_sum)
    VALUES(1, '2011-11-12', '2011-12-12', 1000);
INSERT INTO k_bill
    (contract_num, bill_date, bill_term, bill_sum)
    VALUES(1, '2011-12-12', '2012-01-12', 2000);
INSERT INTO k_bill
     (contract_num, bill_date, bill_term, bill_sum)
     VALUES(1, '2012-01-12', '2012-02-12',2000);
INSERT INTO k_bill
     (contract_num, bill_date, bill_term, bill_sum)
     VALUES(2, '2011-12-12', '2012-01-12', 6000);
INSERT INTO k_bill
     (contract_num, bill_date, bill_term, bill_sum)
     VALUES(2, '2012-01-12', '2012-02-12', 2000);
INSERT INTO k_bill
     (contract_num, bill_date, bill_term, bill_sum)
     VALUES(3, '2012-01-12', '2012-02-12', 2500);
INSERT INTO k_bill
     (contract_num, bill_date, bill_term, bill_sum)
     VALUES(4, '2011-12-12', '2012-01-12', 1500);
INSERT INTO k_bill
     (contract_num, bill_date, bill_term, bill_sum)
     VALUES(5, '2011-12-12', '2012-01-12', 1200);
INSERT INTO k_bill
     (contract_num, bill_date, bill_term, bill_sum)
     VALUES(5, '2012-01-12', '2012-02-12', 10000);
```

Посмотрим результат заполнения, для этого выполним следующую команду:

```
    SELECT * FROM k_bill;
```

```
bill  bill                  bill         bill  contract bill
_num  _date                 _term        _peni _num     _sum
-------------------------------------------------------------
1 2011-11-12 00:00:00.000 2011-12-12 00:00:00.000 0 1 1000
2 2011-12-12 00:00:00.000 2012-01-12 00:00:00.000 0 1 2000
3 2012-01-12 00:00:00.000 2012-02-12 00:00:00.000 0 1 2000
4 2011-12-12 00:00:00.000 2012-01-12 00:00:00.000 0 2 3000
5 2012-01-12 00:00:00.000 2012-02-12 00:00:00.000 0 2 2000
6 2012-01-12 00:00:00.000 2012-02-12 00:00:00.000 0 3 2500
7 2011-12-12 00:00:00.000 2012-01-12 00:00:00.000 0 4 1000
8 2011-12-12 00:00:00.000 2012-01-12 00:00:00.000 0 5 1200
9 2012-01-12 00:00:00.000 2012-02-12 00:00:00.000 0 5 2000

(9 row(s) affected)
```

Таблица "Платежи"

```
INSERT INTO k_payment
    (payment_num, bill_num, payment_date, payment_sum)
    VALUES(1, 1, '2011-12-01', 1000);
INSERT INTO k_payment
    (payment_num, bill_num, payment_date, payment_sum)
    VALUES(1, 2, '2011-12-15', 1000);
INSERT INTO k_payment
    (payment_num, bill_num, payment_date, payment_sum)
    VALUES(1, 3, '2012-01-13', 1500);
INSERT INTO k_payment
    (payment_num, bill_num, payment_date, payment_sum)
    VALUES(2, 3, '2012-01-15', 500);
INSERT INTO k_payment
    (payment_num, bill_num, payment_date, payment_sum)
    VALUES(1, 4, '2012-01-12', 1000);
INSERT INTO k_payment
    (payment_num, bill_num, payment_date, payment_sum)
    VALUES(1, 7, '2012-01-05', 100);
INSERT INTO k_payment
    (payment_num, bill_num, payment_date, payment_sum)
    VALUES(2, 7, '2012-01-12', 900);
INSERT INTO k_payment
    (payment_num, bill_num, payment_date, payment_sum)
    VALUES(1, 8, '2011-12-25', 1000);
```

Посмотрим результат заполнения, для этого выполним следующую команду:

```
SELECT * FROM k_payment;
```

```
payment_num  bill_num  payment_date            payment_sum
-----------------------------------------------------------
1            1         2011-12-01 00:00:00.000  1000.00
1            2         2011-12-15 00:00:00.000  1000.00
1            3         2012-01-13 00:00:00.000  1500.00
1            4         2012-01-12 00:00:00.000  1000.00
1            7         2012-01-05 00:00:00.000   100.00
1            8         2011-12-25 00:00:00.000  1000.00
2            3         2012-01-15 00:00:00.000   500.00
2            7         2012-01-12 00:00:00.000   900.00

(8 row(s) affected)
```

Таблица "Товары/услуги" (или "Прайс-лист")

```
INSERT INTO k_price (price_name, price_sum, type_num)
    VALUES('Материализация духов',1000, 2);
INSERT INTO k_price (price_name, price_sum, type_num)
    VALUES('Раздача слонов',100, 2);
INSERT INTO k_price (price_name, price_sum, type_num)
    VALUES('Слоновий бивень',3000, 1);
INSERT INTO k_price (price_name, price_sum, type_num)
    VALUES('Моржовый клык',1500, 1);
INSERT INTO k_price (price_name, price_sum, type_num)
    VALUES('Копыто Пегаса',5000, 1);
```

Посмотрим результат заполнения, для этого выполним следующую команду:

```
SELECT * FROM k_price;
```

```
price_num  price_name            price_sum  type_num
-----------------------------------------------------------
1          Материализация духов  1000.00     2
2          Раздача слонов         100.00     2
3          Слоновий бивень       3000.00     1
4          Моржовый клык         1500.00     1
5          Копыто Пегаса         5000.00     1

(5 row(s) affected)
```

Таблица "Протоколы счетов"

```
INSERT INTO k_protokol
    (price_num, bill_num, kolvo, price_sum)
    VALUES(1, 1, 1, 1000);
INSERT INTO k_protokol
    (price_num, bill_num, kolvo, price_sum)
    VALUES(1, 2, 2, 1000);
INSERT INTO k_protokol
    (price_num, bill_num, kolvo, price_sum)
    VALUES(2, 3, 20, 100);
INSERT INTO k_protokol
    (price_num, bill_num, kolvo, price_sum)
    VALUES(3, 4, 2, 3000);
INSERT INTO k_protokol
    (price_num, bill_num, kolvo, price_sum)
    VALUES(1, 5, 1, 1000);
INSERT INTO k_protokol
```

```
    (price_num, bill_num, kolvo, price_sum)
    VALUES(2, 5, 10, 100);
INSERT INTO k_protokol
    (price_num, bill_num, kolvo, price_sum)
    VALUES(1, 6, 2, 1000);
INSERT INTO k_protokol
    (price_num, bill_num, kolvo, price_sum)
    VALUES(2, 6, 5, 100);
INSERT INTO k_protokol
    (price_num, bill_num, kolvo, price_sum)
    VALUES(4, 7, 1, 1500);
INSERT INTO k_protokol
    (price_num, bill_num, kolvo, price_sum)
    VALUES(1, 8, 1, 1000);
INSERT INTO k_protokol
    (price_num, bill_num, kolvo, price_sum)
    VALUES(2, 8, 2, 100);
INSERT INTO k_protokol
    (price_num, bill_num, kolvo, price_sum)
    VALUES(5, 9, 2, 5000);
```

Посмотрим результат заполнения, для этого выполним следующую команду:

```
SELECT * FROM k_ protokol;
```

```
price_num  bill_num  kolvo price_sum
-----------------------------------------------------------
1          1         1     1000.00
1          2         2     1000.00
1          5         1     1000.00
1          6         2     1000.00
1          8         1     1000.00
2          3         20     100.00
2          5         10     100.00
2          6         5      100.00
2          8         2      100.00
3          4         2     3000.00
4          7         1     1500.00
5          9         2     5000.00

(12 row(s) affected)
```

Вопрос

Можно ли для таблицы, в которой имеется поле `IDENTITY`, выполнить команду `INSERT`, не указывая явно список полей? Проверьте.

Задание для индивидуальной работы 4

Напишите и отладьте сценарий, который вставляет по несколько строк в каждую таблицу вашей базы данных.

DQL. Запросы

DQL (Data Query Language) – язык запросов, составная часть SQL. Состоит из единственной команды `SELECT`. По поводу этой команды написаны целые книги, здесь мы кратко на примерах рассмотрим ее основные возможности. За более подробной информацией обращайтесь к **MSDN** и справочной литературе.

Обязательные ключевые слова команды – `SELECT` и `FROM`.

Выборка из одной таблицы

Тривиальная выборка всех полей и всех строк одной таблицы.

Получить полную информацию обо всех предприятиях:

```
SELECT * FROM  k_firm
```

Выбор отдельных полей таблицы.

Получить названия и адреса всех предприятий:

```
SELECT firm_name, firm_addr  FROM  k_firm
```

Результат:

```
firm_name firm_addr
----------------------
Альфа     Москва
Бета      Казань
Гамма     Париж
Дельта    Лондон
Омега     Токио
```

Поля выборки можно переименовывать. Если новое название состоит из нескольких слов, помещайте его в двойные кавычки.

```
SELECT    firm_name AS "Название предприятия",
          firm_addr  AS "Адрес предприятия"
     FROM  k_firm
```

Результат:

```
Название предприятия    Адрес предприятия
-------------------------------------------------------------
Альфа                   Москва
Бета                    Казань
Гамма                   Париж
Дельта                  Лондон
Омега                   Токио
```

В списке полей выборки можно использовать выражения. В этом случае часто приходится преобразовывать данные из одного типа в другой с помощью функции CONVERT. Строковые константы следует помещать в одинарные кавычки. Операция + означает сцепление строк.

Распечатать информацию о счетах:

```
SELECT 'Счет № '+CONVERT(CHAR(6),bill_num)+
       ' от '+ CONVERT(CHAR(12),bill_date)+
       ' на сумму '+CONVERT(CHAR(9),bill_sum)
         FROM k_bill
```

Результат:

```
-----------------------------------------------
Счет № 1      от ноя 12 2011  на сумму 1000
Счет № 2      от дек 12 2011  на сумму 2000
Счет № 3      от янв 12 2012  на сумму 2000
Счет № 4      от дек 12 2011  на сумму 6000
Счет № 5      от янв 12 2012  на сумму 2000
Счет № 6      от янв 12 2012  на сумму 2500
Счет № 7      от дек 12 2011  на сумму 1500
Счет № 8      от дек 12 2011  на сумму 1200
Счет № 9      от янв 12 2012  на сумму 10000
```

Для того чтобы исключить дубликаты строк, нужно использовать ключевое слово DISTINCT.

Напечатать список городов, в которых находятся предприятия-клиенты:

```
SELECT DISTINCT firm_addr FROM k_firm
```

Результат:

```
firm_addr
----------
Казань
Лондон
Москва
Париж
Токио
```

Использование условий отбора

Для выбора отдельных строк по некоторому критерию используется ключевое слово WHERE

Получить список предприятий, расположенных в Москве:

```
SELECT    firm_name as "Название предприятия"
```

```
    FROM  k_firm
    WHERE firm_addr='Москва'
```

Результат:

```
Название предприятия
--------------------
Альфа
```

Для сравнения поля со значением `NULL` нельзя использовать операции = и !=, вместо них нужно использовать выражения `IS NULL` и `IS NOT NULL`.

Получить список постоянно работающих сотрудников, т.е., таких, у которых staff_termdate равно NULL:

```
SELECT staff_name FROM k_staff
    WHERE staff_termdate IS NULL
```

Результат:

```
staff_name
-----------
Иванов
Петров
Сидоров
```

Условия могут быть сложные, представляющие собой комбинацию нескольких операций сравнения. В них можно использовать логические связки `AND` и `OR`, а также отрицание `NOT`.

Получить список предприятий, расположенных в Москве или Казани:

```
SELECT  firm_name as "Название предприятия"
     FROM  k_firm
    WHERE firm_addr='Москва' OR firm_addr='Казань'
```

Результат:

```
Название предприятия
--------------------
Альфа
Бета
```

Если условие заключается в сравнении поля со списком значений, удобно использовать ключевое слово `IN`.

Получить список предприятий, расположенных в Москве или Казани:

```
SELECT    firm_name as "Название предприятия"
     FROM  k_firm
WHERE firm_addr IN ('Москва','Казань')
```

Результат:

```
Название предприятия
--------------------
```

```
Альфа
Бета
```

Если условие заключается в сравнении поля с диапазоном значений, удобно использовать ключевое слово `BETWEEN`.

Получить список договоров, заключенных в ноябре 2011 г.:

```
SELECT  * FROM  k_contract
WHERE contract_date BETWEEN '2011-11-01' AND '2011-11-30'
```

Заметим, что полезно предварительно задать желаемый формат даты **год-месяц-день**:

```
SET DATEFORMAT YMD
```

Результат:

```
contract_num contract_date contract_type firm_num staff_num
------------------------------------------------------------
1            2011-11-01 00:00:00.000 A     1        1
4            2011-11-15 00:00:00.000 A     2        2
7            2011-11-12 00:00:00.000 A     4        1
```

Для полей строкового типа можно применять сравнение с подстрокой.

Получить список сотрудников, фамилия которых начинается на И:

```
SELECT  staff_name FROM  k_staff
     WHERE staff_name LIKE 'И%'
```

*(рассмотрите более подробно использование LIKE в **MSDN**)*

Результат:

```
staff_name
----------
Иванов
```

Использование агрегирующих функций

Для подсчета итоговых значений используются функции `SUM`, `COUNT`, `MAX`, `MIN`, `AVG`. Если не используется группировка строк, запрос с использованием итоговой функции вернет *ровно* одну строку.

Подсчитать, на какую сумму выставлены счета в декабре.

```
SELECT  SUM(bill_sum) FROM  k_bill
     WHERE bill_date
          BETWEEN '2011-12-01' AND '2011-12-31'
```

Результат:

```
---------------------------------------
10700
```

Функция `COUNT` позволяет подсчитать, сколько строк в таблице имеется вообще.

Подсчитать количество сотрудников.

```
SELECT COUNT(*) FROM k_staff
```

Результат:

```
-----------
5
```

А также эта функция позволяет подсчитать, сколько строк с не-NULL-значениями в определенном поле.

Подсчитать количество временно работающих сотрудников (у них заполнен срок окончания трудового довора – поле staff_termdate).

```
SELECT COUNT(staff_termdate) FROM k_staff
```

Результат:

```
-----------
0
```

Сортировка

Для сортировки используется ключевое слово `ORDER BY` и имя поля или его номер в списке полей выборки.

Напечатать список сотрудников, отсортированный по алфавиту:

```
SELECT staff_name FROM k_staff ORDER BY 1
```

Результат:

```
staff_name
-------------
Григорьев
Иванов
Петров
Семенов
Сидоров
```

Можно сортировать строки даже по такому полю, которое не входит в список полей выборки.

Напечатать список сотрудников, отсортированный по дате поступления на работу:

```
SELECT staff_name FROM k_staff ORDER BY staff_hiredate
```

Сортировать данные можно и по убыванию. Кроме того, можно ограничить количество строк в результате.

Напечатать информацию о 5 последних выписанных счетах в порядке убывания даты счета:

```
SELECT TOP 5 bill_num, bill_date
FROM k_bill ORDER BY bill_date DESC
```

Результат:

```
bill_num bill_date
-------- -----------------------
3        2012-01-12 00:00:00.000
6        2012-01-12 00:00:00.000
5        2012-01-12 00:00:00.000
9        2012-01-12 00:00:00.000
4        2011-12-12 00:00:00.000
```

Подзапросы

Для более сложных формулировок иногда удобно использовать подзапросы.

Подзапрос всегда указывается в скобках.

Подзапрос может быть *несвязанным,* т.е. в формулировке подзапроса нет ссылки на главный запрос. В этом случае подзапрос выполняется один раз при выполнении главного запроса. В данном примере используется ключевое слово `IN`, так как подзапрос может возвращать несколько значений.

Получить список договоров, по которым в декабре выписаны счета:

```
SELECT contract_num, contract_date FROM k_contract
     WHERE contract_num IN
          (SELECT contract_num FROM k_bill
          WHERE bill_date
          BETWEEN '2011-12-01' AND '2011-12-31')
```

Результат:

```
contract_num contract_date
---------------------------------
1            2011-11-01 00:00:00.000
2            2011-10-01 00:00:00.000
4            2011-11-15 00:00:00.000
5            2011-08-01 00:00:00.000
```

Тот же самый запрос с использованием ключевого слова `ANY`:

```
SELECT contract_num, contract_date FROM k_contract c
     WHERE contract_num = ANY
```

```
          (SELECT contract_num FROM k_bill
          WHERE bill_date
               BETWEEN '2011-12-01' AND '2011-12-31')
```

Тот же самый запрос можно выполнить и с помощью *связанного* подзапроса, т.е., подзапроса, в котором есть ссылка на главный запрос. Для ссылки на таблицу главного запроса нужно указать псевдоним. Такой подзапрос будет выполняться заново для каждой строки главного запроса.

Кроме того, в данном примере иллюстрируется использование ключевого слова `EXISTS`:

```
SELECT contract_num, contract_date FROM k_contract c
     WHERE EXISTS
          (SELECT * FROM k_bill b
          WHERE bill_date
               BETWEEN '2011-12-01' AND '2011-12-31'
               AND c.contract_num=b.contract_num)
```

Пример использования ключевого слова `ALL`.

Напечатать информацию о товаре (товарах) с наименьшей ценой.

```
SELECT price_name, price_sum FROM k_price
     WHERE price_sum <= ALL
          (SELECT price_sum FROM k_price)
```

Результат:

```
price_name      price_sum
-------------------------
Раздача слонов  100.00
```

Этот запрос можно сформулировать и по-другому. В этом примере мы можем использовать операцию сравнения =, т.к. подзапрос возвращает ровно одну строку и один столбец.

```
SELECT price_name, price_sum FROM k_price
     WHERE price_sum =
          (SELECT MIN(price_sum) FROM k_price)
```

А так, как в следующем примере, запрос формулировать нельзя, поскольку при запуске будет выдана ошибка. Если используются агрегирующие функции *без* группировки, в списке полей могут присутствовать *только* агрегирующие функции.

```
SELECT price_name, MIN(price_sum) FROM k_price
```

Результат – сообщение об ошибке:

```
Msg 8120, Level 16, State 1, Line 1
```

```
Столбец "k_price.price_name" недопустим в списке выбора,
поскольку он не содержится ни в статистической функции, ни в
предложении GROUP BY.
```

(Подробнее по поводу ошибок – см. Приложение 1.)

Группировка

Для подведения итога по группе данных используется комбинация ключевого слова `GROUP BY` и агрегирующих функций. Причем в списке полей для выборки могут присутствовать *только* поля группировки и агрегирующие функции (при необходимости можно просто **расширить** список полей группировки).

Получить список договоров и общую сумму счетов по каждому договору:

```
SELECT contract_num, SUM(bill_sum) AS contract_sum
     FROM k_bill
     GROUP BY contract_num
```

Результат:

```
contract_num contract_sum
-------------------------
1            5000
2            8000
3            2500
4            1500
5            11200
```

В том случае, когда нужно выбрать не все группы, а только некоторые из них, используется ключевое слово `HAVING`:

Получить список договоров, имеющих 2 или более счетов, и общую сумму счетов по каждому договору:

```
SELECT contract_num, SUM(bill_sum) AS contract_sum
     FROM k_bill
     GROUP BY contract_num
     HAVING COUNT(bill_num)>=2
```

Результат:

```
contract_num contract_sum
------------ ----------------------------------------
1            5000
2            8000
5            11200
```

Для связи таблиц можно использовать то же ключевое слово WHERE, как и для условий отбора. При выборке из нескольких таблиц рекомендуется всегда использовать псевдонимы таблиц. Дело в том, что если в разных таблицах имеются одинаковые поля, то всегда нужно уточнять, к какой таблице они относятся, т.е., использовать синтаксис имя_таблицы.имя_поля. А так как имена таблиц обычно длинные, удобно заменять их псевдонимами.

Напечатать список договоров с указанием названия предприятия.

```
SELECT firm_name, contract_num, contract_date
     FROM k_firm f, k_contract c
     WHERE f.firm_num=c.firm_num
```

Результат:

```
firm_name contract_num contract_date
-------------------------------------------------------
Альфа     1            2011-11-01 00:00:00.000
Альфа     2            2011-10-01 00:00:00.000
Альфа     3            2011-09-01 00:00:00.000
Бета      4            2011-11-15 00:00:00.000
Бета      5            2011-08-01 00:00:00.000
Гамма     6            2011-07-15 00:00:00.000
Дельта    7            2011-11-12 00:00:00.000
```

То же самое можно получить, если использовать синтаксис JOIN...ON. Это так называемое *внутреннее* (INNER) соединение. Строки соединяются, если совпадают значения полей в условии ON.

```
SELECT firm_name, contract_num, contract_date
     FROM k_firm f JOIN k_contract c ON f.firm_num=c.firm_num
```

Кроме внутреннего, бывают еще левое (LEFT), правое (RIGHT) и полное (FULL) соединения.

Рассмотрим, например, левое соединение. В результат попадут строки, в которых совпадают значения полей в условии ON, и те строки из левой таблицы, для которых не нашлось соответствующих строк в правой таблице. Поля из правой таблицы будут заполнены значениями NULL.

Напечатать список договоров с указанием названия предприятия плюс список предприятий, у которых нет договоров:

```
SELECT firm_name, contract_num, contract_date
     FROM k_firm f LEFT JOIN k_contract c ON
     f.firm_num=c.firm_num
```

Результат:

```
firm_name contract_num contract_date
-----------------------------------------------------------
Альфа     1            2011-11-01 00:00:00.000
Альфа     2            2011-10-01 00:00:00.000
Альфа     3            2011-09-01 00:00:00.000
Бета      4            2011-11-15 00:00:00.000
Бета      5            2011-08-01 00:00:00.000
Гамма     6            2011-07-15 00:00:00.000
Дельта    7            2011-11-12 00:00:00.000
Омега     NULL         NULL
```

А что будет в том случае, если условие связи вообще не указывать? Получится так называемое *декартово произведение* таблиц, в котором *каждая* строка первой таблицы будет сцеплена с *каждой* строкой второй таблицы. Результат получается обычно очень большим и не имеющим смысла.

```
SELECT firm_name, contract_num, contract_date
     FROM k_firm f, k_contract c
```

Разумеется, можно связывать не только две, а три и более таблицы, использовать в этих запросах подзапросы, группировки и т.п. Например:

Напечатать информацию о платежах с указанием названия предприятия:

```
SELECT firm_name, payment_date, payment_sum
 FROM k_firm f, k_contract c, k_bill b, k_payment p
 WHERE f.firm_num=c.firm_num  AND
  c.contract_num=b.contract_num AND b.bill_num=p.bill_num
```

Результат:

```
firm_name    payment_date             payment_sum
-----------------------------------------------------------
Альфа        2011-12-01 00:00:00.000   1000.00
Альфа        2011-12-15 00:00:00.000   1000.00
Альфа        2012-01-13 00:00:00.000   1500.00
Альфа        2012-01-12 00:00:00.000   1000.00
Бета         2012-01-05 00:00:00.000    100.00
Бета         2011-12-25 00:00:00.000   1000.00
Альфа        2012-01-15 00:00:00.000    500.00
Бета         2012-01-12 00:00:00.000    900.00
```

Объединение запросов

Для объединения результатов двух и более запросов нужно использовать ключевое слово `UNION`. Объединяемые запросы должны

иметь одинаковое количество и тип полей. Параметр `ORDER BY` , если он нужен, следует указывать только в *последнем* запросе.

Получить список договоров и общую сумму счетов по каждому договору, а также строку с итоговой суммой:

```
SELECT 'Договор № '+CONVERT(CHAR(6),contract_num)+
          'на сумму ' AS "Номер",
     SUM(bill_sum) AS "Сумма" FROM k_bill
          GROUP BY contract_num
UNION
SELECT 'ИТОГО: ', SUM(bill_sum) FROM k_bill ORDER BY 1
```

Результат:

```
Номер                       Сумма
--------------------------- ------------
Договор № 1     на сумму  5000
Договор № 2     на сумму  8000
Договор № 3     на сумму  2500
Договор № 4     на сумму  1500
Договор № 5     на сумму  11200
ИТОГО:                    28200
```

И еще несколько примеров

Получить прайс-лист с суммой заказов по каждому товару. Обратите внимание, что название и цена товара указываются в списке полей для группировки только для того, чтобы их можно было использовать в списке полей выборки. Для группировки здесь достаточно номера товара, так как он уникальный.

```
SELECT pr.price_name, pr.price_sum,
     SUM(prot.kolvo*prot.price_sum)
FROM k_price pr, k_protokol prot
     WHERE pr.price_num=prot.price_num
    GROUP BY pr.price_num, pr.price_name, pr.price_sum
```

Результат:

```
price_name              price_sum
-------------------------------------------------------
Материализация духов   1000.00     7000.00
Раздача слонов         100.00      3700.00
Слоновий бивень        3000.00     6000.00
Моржовый клык          1500.00     1500.00
Копыто Пегаса          5000.00    10000.00
```

Полностью оплаченные счета:

```
SELECT    b.bill_num AS "Номер счета",
     b.bill_date AS "Дата счета",
```

```
     b.bill_sum AS "Сумма счета",
     SUM(p.payment_sum) AS "Сумма оплаты"
FROM k_bill b, k_payment p
WHERE b.bill_num=p.bill_num AND
     b.bill_sum<=
          (SELECT SUM(payment_sum) FROM k_payment p2
          WHERE b.bill_num=p2.bill_num)
GROUP BY b.bill_num, b.bill_date, b.bill_sum
```

Результат:

```
Номер счета    Дата счета         Сумма счета Сумма оплаты
-----------------------------------------------------------
1             2011-11-12 00:00:00.000   1000  1000.00
3             2012-01-12 00:00:00.000   2000  2000.00
```

Полностью неоплаченные счета

```
SELECT    b.bill_num AS "Номер счета",
     b.bill_date AS "Дата счета",
     b.bill_sum AS "Сумма счета",
     0 AS "Сумма оплаты"
FROM k_bill b
WHERE b.bill_num NOT IN (SELECT bill_num FROM k_payment)
```

Результат:

```
Номер счета    Дата счета         Сумма счета Сумма оплаты
-----------------------------------------------------------
5       2012-01-12 00:00:00.000    2000        0
6       2012-01-12 00:00:00.000    2500        0
9       2012-01-12 00:00:00.000   10000        0
```

Частично оплаченные счета – обратите внимание, что в этом примере в параметре FROM вместо второй таблицы используется вложенный SELECT

```
SELECT    b.bill_num AS "Номер счета",
     b.bill_date AS "Дата счета",
     b.bill_sum AS "Сумма счета",
     p.pay_sum AS "Сумма оплаты"
FROM k_bill b,
     (SELECT bill_num, SUM(payment_sum) as pay_sum
          FROM k_payment
          GROUP BY bill_num) p
WHERE b.bill_sum >p.pay_sum AND b.bill_num=p.bill_num
```

Результат:

```
Номер счета    Дата счета         Сумма счета Сумма оплаты
-----------------------------------------------------------
2            2011-12-12 00:00:00.000   2000   1000.00
4            2011-12-12 00:00:00.000   6000   1000.00
```

```
7            2011-12-12 00:00:00.000   1500   1000.00
8            2011-12-12 00:00:00.000   1200   1000.00
```

Вопрос

Какие функции есть в языке SQL server? Изучите данный вопрос самостоятельно по **MSDN** или другим источникам.

Задание для индивидуальной работы 5

Напишите несколько (не менее 5) интересных запросов к вашей базе данных. Используйте вложенные подзапросы, группировки, итоговые значения, выборки из нескольких таблиц. Если ваш запрос требует ввода параметра, замените его пока на константу, запросы с параметрами можно будет в дальнейшем реализовать с помощью хранимых процедур.

DDL. Представления

Представления – это объекты базы данных, которые можно рассматривать как виртуальные таблицы. На самом деле хранится только формулировка команды `SELECT`, с помощью которой производится выборка данных из реальных таблиц.

Необходимость в использовании представлений возникает, например, в том случае, когда нужно запретить доступ пользователя к отдельным столбцам или строкам таблицы – тогда можно просто написать представление, в котором эти столбцы или строки не будут присутствовать, и предоставить доступ пользователю именно к этому представлению, а не к реальной таблице.

Другой полезной возможностью является вычисление значений, которые не хранятся непосредственно в таблице, но всегда могут быть рассчитаны.

Представление, как и запрос, может содержать информацию из разных таблиц.

Представления могут быть *обновляемыми* (т.е., предоставлять возможность не только чтения, но и изменения данных в исходных таблицах) и *необновляемыми*. Представление будет обновляемым только в том случае, если его структура такова, что SQL server может точно определить, в какие строки каких таблиц нужно поместить измененные данные. Необновляемыми будут, например, представления, содержащие итоговые данные и группировки.

Для создания представлений используется команда `CREATE VIEW`.

Краткий формат этой команды:

```
CREATE VIEW имя_представления AS
     Команда_SELECT
```

Команду создания представления нужно либо выполнять отдельно от других команд, либо сразу после нее поставить команду `GO`, как в следующем примере.

Например, создадим представление, содержащее список договоров и их кураторов для отдела с номером 1. *Будет ли это представление обновляемым?*

```
CREATE VIEW k_contract1
AS
SELECT k_contract.contract_num, k_contract.contract_date,
    k_contract.contract_type, k_contract.firm_num,
```

```
    k_staff.staff_name
FROM k_contract INNER JOIN
    k_staff ON k_contract.staff_num = k_staff.staff_num
WHERE dept_num = 1

GO
```

Для просмотра представления следует выполнить команду

```
SELECT * FROM k_contract1
```

Результат выполнения команды:

```
contract_num contract_date contract_type firm_num staff_name
------------------------------------------------------------
1            2011-11-01 00:00:00.000  A   1   Иванов
3            2011-09-01 00:00:00.000  C   1   Иванов
6            2011-07-15 00:00:00.000  C   3   Иванов
7            2011-11-12 00:00:00.000  A   4   Иванов
```

Создадим вспомогательное представление для запросов о полностью оплаченных и частично оплаченных счетах (см. предыдущее занятие). Это представление для каждого счета содержит его номер и сумму оплаты.

```
CREATE VIEW k_pay_sum
AS
SELECT bill_num, SUM(payment_sum) AS pay_sum
FROM k_payment
GROUP BY bill_num

GO
```

Для просмотра представления следует выполнить команду

```
SELECT * FROM k_pay_sum.
```

Это представление не будет обновляемым.

Результат выполнения команды:

```
bill_num pay_sum
----------------------
1        1000.00
2        1000.00
3        2000.00
4        1000.00
7        1000.00
8        1000.00
```

Теперь с помощью данного представления можно переформулировать сам запрос, он станет проще:

Полностью оплаченные счета

```
SELECT    b.bill_num AS "Номер счета",
     b.bill_date AS "Дата счета",
     b.bill_sum AS "Сумма счета",
     p.pay_sum AS "Сумма оплаты"
FROM k_bill b, k_pay_sum p
WHERE b.bill_num=p.bill_num AND
     b.bill_sum<=p.pay_sum
```

Задание для индивидуальной работы 6

Создайте несколько (не менее 3) представлений для вашей базы данных. Будут ли они обновляемыми или нет? Проверьте.

Хранимые процедуры

Хранимые процедуры – это объекты базы данных, которые представляют собой программы, манипулирующие данными и **выполняемые на сервере**. Эти программы, кроме команд языка SQL, могут использовать немногочисленные управляющие команды.

Структура хранимой процедуры следующая:

```
CREATE PROC[EDURE] имя_процедуры [параметры]
     AS
Код процедуры
```

Локальные переменные и параметры в процедуре начинаются с символа @.

Глобальные переменные начинаются с символов @@. Есть довольно много системных глобальных переменных с полезной информацией. Некоторые из них мы будем использовать в следующих темах.

Объявление переменных имеет вид

```
DECLARE имя_переменной тип_переменной [(длина)]
```

Блок операторов заключается в команды `BEGIN … END`

Оператор присвоения выгладит довольно странно:

```
SELECT переменная=значение
```

Зато с помощью такого синтаксиса при выполнении команды SELECT можно сохранять значения в переменных.

Альтернативный формат оператора присвоения:

```
SET переменная=значение
```

Условный оператор выглядит так:

```
IF условие
     Оператор1
[ELSE
     Оператор2]
```

Цикл по счетчику отсутствует, есть только цикл по условию

```
WHILE условие
     Оператор
```

Для прерывания цикла используется команда `BREAK`.

Для прерывания итерации цикла используется команда `CONTINUE`.

Оператор печати имеет вид `PRINT выражение`

Выход из процедуры: `RETURN [код_завершения]`

Это первый способ возвращения значения из процедуры – таким образом можно возвращать только целочисленное значение.

Команда

```
RAISERROR сообщение, уровень_опасности, код_состояния
```

применяется для вывода сообщений об ошибках и прочих предупреждений в стандартной для SQL server форме.

Выражение `CASE` применяется для выбора на основании нескольких опций:

```
CASE выражение
     WHEN вариант1 THEN выражение1
     WHEN вариант2 THEN выражение2
     …
     ELSE выражениеN
END
```

Создадим процедуру, которая в качестве параметра получает фамилию сотрудника и печатает список всех договоров, которые он курирует.

Это второй способ возвращения значений из процедуры – печать результата выполнения команды SELECT.

```
CREATE PROCEDURE show_contracts @name Varchar(30)
AS
SELECT contract_num, contract_date, contract_type
     FROM k_contract c JOIN k_staff s ON
     c.staff_num=s.staff_num
     WHERE s.staff_name=@name
GO
```

Для запуска этой процедуры нужно выполнить, например, команду

```
EXEC show_contracts 'Иванов'
```

В результате получим:

```
contract_num contract_date              contract_type
------------ -------------------------- -------------
1            2011-11-01 00:00:00.000     A
3            2011-09-01 00:00:00.000     C
6            2011-07-15 00:00:00.000     C
7            2011-11-12 00:00:00.000     A
```

Создадим процедуру «Распродажа», которая находит самый непродаваемый (по количеству) товар и уценивает его на заданный процент (по умолчанию задается 10 процентов).

```
CREATE PROCEDURE clearance @percent Int = 10
AS
DECLARE @p Int
IF @percent > 0 AND @percent < 100
```

```
     BEGIN
          SELECT @p=price_num FROM k_protokol
               GROUP BY price_num
               HAVING SUM(kolvo)<=ALL
                    (SELECT SUM(kolvo) FROM k_protokol
                         GROUP BY price_num)
          UPDATE k_price
               SET price_sum=price_sum*(100-@percent)/100
               WHERE price_num=@p
     END
GO
```

Содержимое таблицы "Прайс-лист" до выполнения процедуры:

```
price_num price_name            price_sum
-----------------------------------------------
1         Материализация духов  1000.00
2         Раздача слонов        100.00
3         Слоновий бивень       3000.00
4         Моржовый клык         1500.00
5         Копыто Пегаса         5000.00
```

Для запуска этой процедуры нужно выполнить, например, команду

```
EXEC clearance 10
```

Содержимое таблицы "Прайс-лист" после выполнения процедуры:

```
price_num price_name            price_sum
---------------------------------------
1         Материализация духов  1000.00
2         Раздача слонов        100.00
3         Слоновий бивень       3000.00
4         Моржовый клык         1350.00
5         Копыто Пегаса         5000.00
```

Как видим, товар с номером 4 в прайс-листе уценен на 10%.

В том случае, если из хранимой процедуры нужно вернуть значение переменной, нужно объявить эту переменную как выходной (OUTPUT) параметр процедуры в двух местах: в описании процедуры и в вызове процедуры. *Это третий способ возвращения значений из процедуры – выходные параметры.*

Пусть, например, в предыдущей процедуре мы хотим не только уценить товар, но и возвратить его номер. Описание процедуры будет выглядеть следующим образом:

```
CREATE PROCEDURE clearance @percent Int, @p Int OUTPUT
AS
IF @percent > 0 AND @percent < 100
       BEGIN
              SELECT @p=price_num FROM k_protokol
                 GROUP BY price_num
                 HAVING SUM(kolvo)<=ALL
                      (SELECT SUM(kolvo) FROM k_protokol
```

```
                                GROUP BY price_num)
            UPDATE k_price
               SET price_sum=price_sum*(100-@percent)/100
                   WHERE price_num=@p
      END
```

А вызов процедуры будет выглядеть следующим образом (все три команды должны выполняться вместе):

```
DECLARE @num NUMERIC(6)
EXEC clearance 1, @num OUTPUT
PRINT 'Уценили товар с номером '+STR(@num)
```

И еще один пример. Предположим, у нас есть таблица для хранения списка счетов:

```
CREATE TABLE bill_list
(name VARCHAR(20), dat DATETIME, summa NUMERIC(9,2))
```

Мы хотим сформировать список выставленных за месяц счетов с названиями предприятий и с итогами по дням. Рассмотрите этот пример самостоятельно. Месяц и год передаются в процедуру в качестве параметров. Функция `DATEDIFF` здесь вычисляет разность между двумя датами в днях.

```
CREATE PROCEDURE calc_bill_list @mon Int, @year Int
AS
DECLARE @day Int, @end Int, @date DateTime
SET @day=1
IF @mon=2
       IF @year%4=0
              SET @end=29
       ELSE
              SET @end=28
ELSE   IF @mon=4 OR @mon=6 OR @mon=9 OR @mon=11
              SET @end=30
       ELSE
              SET @end=31

DELETE FROM bill_list
WHILE (@day<=@end)
  BEGIN
       SET @date=CONVERT(CHAR(2),@mon)+'/'+
              CONVERT(CHAR(2),@day)+'/'+
              CONVERT(CHAR(4),@year)
       INSERT INTO bill_list (name, dat, summa)
         SELECT firm_name, bill_date, bill_sum
              FROM k_firm, k_contract, k_bill
              WHERE  k_firm.firm_num=k_contract.firm_num
              AND
              k_contract.contract_num=k_bill.contract_num
              AND
              DATEDIFF(day, k_bill.bill_date, @date)=0
```

```
      INSERT INTO bill_list (name, dat, summa)
        SELECT '      ИТОГО ЗА:',  @date,
             ISNULL(SUM(bill_sum),0) FROM k_bill
             WHERE
             DATEDIFF(day, k_bill.bill_date, @date)=0
      SET @day=@day+1
  END
```

Обратите внимание на формат команды `INSERT`. В таблицу `bill_list` добавляются строки, являющиеся результатом выполнения команды `SELECT`.

Для удаления хранимой процедуры используется команда:

```
DROP PROCEDURE имя_процедуры
```

Задание для индивидуальной работы 7

Создайте несколько хранимых процедур для вашей базы данных. Можете использовать запросы с параметрами из позапрошлого занятия.

CCL. Курсоры

CCL (Cursor Control language) – язык управления курсорами, составная часть SQL.

Как вы уже поняли, команды манипулирования данными `SELECT`, `UPDATE`, `DELETE` работают сразу с **группами** строк. Эти группы, вплоть до отдельных строк, можно выбрать с помощью опции `WHERE`. Что же делать в том случае, если требуется перебрать строки некоторой таблицы последовательно, одну за другой? Для этого в языке SQL существует такое понятие, как *курсор*. *Курсор* (**cur**rent **s**et of **r**ecord) – это временный набор строк, которые можно перебирать последовательно, с первой до последней.

Для работы с курсорами существуют следующие команды.

Объявление курсора:

```
DECLARE имя_курсора CURSOR FOR SELECT текст_запроса
```

Таким образом, любой курсор создается на основе некоторого оператора `SELECT`.

Открытие курсора:

```
OPEN имя_курсора
```

Только после открытия курсора он становится активным, и из него можно читать строки.

Чтение значений из следующей строки курсора в набор переменных:

```
FETCH имя_курсора INTO список_переменных
```

Переменные в списке должны иметь то же количество и тип, что и столбцы курсора.

Глобальная переменная `@@FETCH_STATUS` принимает ненулевое значение, если строк в курсоре больше нет. Если же набор строк еще не исчерпан, то `@@FETCH_STATUS` равна нулю, и оператор `FETCH` при выполнении перепишет значения полей из текущей строки в переменные.

Закрытие курсора:

```
CLOSE имя_курсора
```

Для **удаления курсора из памяти** используется команда

```
DEALLOCATE имя_курсора
```

Для иллюстрации использования курсора создадим процедуру, начисляющую пени по тем неоплаченным счетам, по которым истек срок платежа.

```
CREATE PROCEDURE peni @percent NUMERIC(5,2)
AS
DECLARE   @num INT,
          @dat DATETIME,
          @days INT,
          @sum NUMERIC(6)

IF @percent > 0 AND @percent < 100

BEGIN
DECLARE cur1 CURSOR FOR
     SELECT bill_num, bill_term FROM k_bill b
          WHERE bill_term<GETDATE()
          AND ( bill_sum>
          (SELECT SUM(payment_sum) FROM k_payment p
               WHERE b.bill_num=p.bill_num)
          OR NOT EXISTS
          (SELECT bill_num FROM k_payment p
               WHERE b.bill_num=p.bill_num)
)

OPEN cur1
FETCH cur1 INTO @num, @dat

WHILE @@FETCH_STATUS=0
BEGIN
     SELECT @days=DATEDIFF(day, @dat, GETDATE())

     SELECT @sum=ISNULL(SUM(payment_sum),0) FROM k_payment
          WHERE @num=bill_num

     UPDATE k_bill
          SET bill_peni=(bill_sum-@sum)*@percent/100*@days
          WHERE @num=bill_num

     FETCH cur1 INTO @num, @dat
END
DEALLOCATE cur1

END
GO
```

Рассмотрим эту процедуру более подробно.

Параметром этой процедуры является процент для вычисления пени.

Объявляем курсор на основе следующего запроса: выбрать счета, которые оплачены не полностью и по которым истек срок платежа (т.е, срок оплаты менее текущей даты). В эту выборку попадут частично оплаченные счета, для которых выполняется условие

```
bill_sum>
     (SELECT SUM(payment_sum) FROM k_payment p
          WHERE b.bill_num=p.bill_num)
```

а также полностью неоплаченные счета, для которых платежей вообще не существует, т.е., выполняется условие

```
NOT EXISTS
     (SELECT bill_num FROM k_payment p
          WHERE b.bill_num=p.bill_num)
```

Открываем курсор и читаем из него в цикле последовательно по одной строке. Каждая строка содержит информацию об одном просроченном неоплаченном счете.

Для текущего счета вычисляем количество дней, на который он просрочен, с помощью функции `DATEDIFF`:

```
DATEDIFF(day, @dat, GETDATE())
```

Первый параметр этой функции означает единицу измерения (дни), второй и третий – даты, для которых мы вычисляем разность (текущая дата минус дата счета).

Далее для текущего счета вычисляем общую сумму пени. Пени будут начисляться на неоплаченную часть суммы счета, по заданному проценту за каждый день просрочки.

Напечатаем список счетов до выполнения процедуры:

```
SELECT bill_num, bill_term, bill_peni, bill_sum
     FROM k_bill
```

```
bill_num bill_term                    bill_peni bill_sum
-------------------------------------------------------
1        2011-12-12 00:00:00.000      0         1000
2        2012-01-12 00:00:00.000      0         2000
3        2012-02-12 00:00:00.000      0         2000
4        2012-01-12 00:00:00.000      0         6000
5        2012-02-12 00:00:00.000      0         2000
6        2012-02-12 00:00:00.000      0         2500
7        2012-01-12 00:00:00.000      0         1500
8        2012-01-12 00:00:00.000      0         1200
9        2012-02-12 00:00:00.000      0        10000
```

Пусть, например, сегодня 1 мая 2012 г. Запустим процедуру.

```
EXEC peni 0.5
```

Напечатаем список счетов после выполнения процедуры:

```
SELECT bill_num, bill_term, bill_peni, bill_sum
     FROM k_bill
```

```
bill_num bill_term                  bill_peni bill_sum
-----------------------------------------------------
1        2011-12-12 00:00:00.000        0       1000
2        2012-01-12 00:00:00.000      550       2000
3        2012-02-12 00:00:00.000        0       2000
4        2012-01-12 00:00:00.000     2750       6000
5        2012-02-12 00:00:00.000      790       2000
6        2012-02-12 00:00:00.000      988       2500
7        2012-01-12 00:00:00.000      275       1500
8        2012-01-12 00:00:00.000      110       1200
9        2012-02-12 00:00:00.000     3950       10000
```

Задание для индивидуальной работы 8

Создайте хранимую процедуру с использованием курсора для вашей базы данных.

Триггеры

Триггеры – это хранимые процедуры специального вида, которые автоматически выполняются при изменении таблицы с помощью операторов `INSERT, UPDATE` и `DELETE`. Триггер создается для определенной таблицы, но может использовать данные других таблиц и объекты других баз данных.

Существует 3 типа триггеров: `INSERT, UPDATE` и `DELETE`. Правила работы с триггерами следующие:

- триггеры запускаются только после выполнения вызвавшего их оператора;
- если при выполнении оператора возникает нарушение какого-либо ограничения или другая ошибка, триггер не срабатывает (даже не начинает выполняться);
- триггер и вызвавший его оператор образует *транзакцию.* Если нужно из триггера отменить вызвавшую его операцию, следует выполнить *откат транзакции* ROLLBACK;
- триггер срабатывает один раз для каждого оператора, независимо от количества изменяемых им записей.

Краткий формат триггера (более подробно смотрите в MSDN):

```
CREATE TRIGGER имя_триггера
     ON имя_таблицы
FOR INSERT | UPDATE | DELETE
     AS
Код_триггера
```

Рассмотрим элементарный пример: при обновлении таблицы «Сотрудники» печатается сообщение. (Не делайте подобных триггеров в качестве задания для самостоятельной работы!)

```
CREATE TRIGGER upd_staff
    ON k_staff FOR UPDATE
    AS
    PRINT 'Обновили таблицу Сотрудники'
```

После создания триггера нужно протестировать его, выполнив команду `UPDATE` для таблицы Сотрудники.

При *добавлении* строки в таблицу ее копия помещается во временную таблицу с именем `Inserted`, при *удалении* – с именем `Deleted`. При *обновлении* старая версия строки помещается во

временную таблицу с именем `Deleted`, новая – с именем `Inserted`. Эти временные таблицы часто используются в триггерах.

Рассмотрим пример триггера вставки, который вызывается при выполнении команды `INSERT` в таблице протоколов счетов. При добавлении новой позиции в счете нам нужно заново пересчитать его общую сумму.

```
CREATE TRIGGER ins_prot
    ON k_protokol FOR INSERT
    AS
    DECLARE @s_new NUMERIC(9,2),
          @kolvo NUMERIC(6),
          @bill_num NUMERIC(6)
     SELECT @kolvo=kolvo FROM Inserted
     IF @kolvo>0
          BEGIN
          SELECT @s_new=p.price_sum,
               @bill_num=bill_num
               FROM k_price p, Inserted i
                    WHERE p.price_num=i.price_num
               IF @s_new !=0
                    UPDATE k_bill
                    SET bill_sum=bill_sum+@s_new*@kolvo
                         WHERE k_bill.bill_num=@bill_num
          END
```

Для тестирования триггера следует выполнить команду добавления, например:

Выберем информацию о счете №1:

```
SELECT bill_num, bill_sum FROM k_bill WHERE bill_num=1
```

Получим:

```
bill_num bill_sum
-----------------
1        1000
```

Теперь добавим строку в протокол этого счета:

```
INSERT INTO k_protokol
     (price_num, bill_num, kolvo, price_sum)
         VALUES(5, 1, 1, 5000);
```

Снова выберем информацию о счете №1:

```
SELECT bill_num, bill_sum FROM k_bill WHERE bill_num=1
```

Получим:

```
bill_num bill_sum
-----------------
1        6000
```

Как видим, сумма счета увеличилась на стоимость выбранного товара.

Рассмотрим пример триггера удаления, который вызывается при выполнении команды DELETE в таблице протоколов счетов. При удалении позиции в счете нам нужно пересчитать его сумму. Здесь возникает следующая проблема – если в команде DELETE было удалено сразу несколько строк, трудно будет их обработать. Поэтому сначала мы выполняем проверку: сколько строк было удалено. Эта информация хранится в глобальной переменной @@ROWCOUNT. Если количество удаленных строк больше 1, выводим сообщение об ошибке и отменяем команду DELETE. В остальном этот триггер похож на предыдущий.

```
CREATE TRIGGER del_prot
    ON k_protokol FOR DELETE
    AS
    DECLARE @s_old NUMERIC(9,2),
@kolvo NUMERIC(6),
@bill_num NUMERIC(6)
    IF @@ROWCOUNT>1
     BEGIN
          RAISERROR
          ('Нельзя удалять более 1 строки за раз!', 16, 1)
          ROLLBACK TRAN
     END
    ELSE
     BEGIN
          SELECT @kolvo=kolvo FROM Deleted
          IF @kolvo>0
          BEGIN
               SELECT @s_old=p.price_sum,
               @bill_num=bill_num
               FROM k_price p, Deleted d
                    WHERE p.price_num=d.price_num
               IF @s_old !=0
                    UPDATE k_bill
                    SET bill_sum=bill_sum-@s_old*@kolvo
                    WHERE k_bill.bill_num=@bill_num
          END
     END
```

Ту же задачу можно решить другим образом, не ограничивая количество удаляемых счетов. Просто пересчитаем суммы для всех счетов. Если в таблице Deleted есть строки протокола для какого-то

счета, его сумма будет уменьшена. Этот триггер получится гораздо короче, но он неэффективен, так как обрабатывает все счета.

Обратите внимание, что в команде UPDATE используется связанный подзапрос.

```
CREATE TRIGGER del_prot
    ON k_protokol FOR DELETE
    AS
    UPDATE k_bill SET bill_sum = bill_sum -
          (SELECT SUM(price_sum*kolvo)
               FROM Deleted d
               WHERE d.bill_num=k_bill.bill_num)
```

Выполним команду создания триггера. Все нормально, ошибок нет.

Теперь попробуем удалить какую-нибудь строку из протокола счетов. Выдается ошибка

```
Cannot insert the value NULL into column 'bill_sum'
```

В чем же дело? Дело в том, что функция SUM вместо ожидаемых числовых значений 0 возвратила NULL-значения для тех счетов, информации о которых нет в таблице Deleted. Чтобы преобразовать ненужные NULL в числовые нули, удобно использовать функцию ISNULL. Она имеет формат

```
ISNULL(выражение, значение_вместо_NULL)
```

В том случае, если выражение не равно NULL, функция возвращает выражение. Если оно равно NULL, то значение_вместо_NULL. Триггер примет вид:

```
CREATE TRIGGER del_prot
    ON k_protokol FOR DELETE
    AS
    UPDATE k_bill SET bill_sum = bill_sum -
          ISNULL((SELECT SUM(price_sum*kolvo)
               FROM Deleted d
               WHERE d.bill_num=k_bill.bill_num),0)
```

Рассмотрим еще один пример. В таблице платежей мы (на свою голову) установили составной первичный ключ: "номер_счета, номер_платежа", причем номер_платежа должен быть уникальным только в пределах его счета. Т.о., мы не могли для заполнения этого поля использовать свойство IDENTITY (по умолчанию в этом поле мы назначили 0). Попробуем создать триггер для поиска максимального

кода платежа по данному счету и формирования нового номера платежа. Все команды в этом триггере вам уже знакомы.

```
CREATE TRIGGER ins_pay
    ON k_payment FOR INSERT
    AS
    DECLARE @n NUMERIC(6),
          @bill NUMERIC(6)
    SELECT @bill=bill_num FROM Inserted
    SELECT @n=ISNULL(MAX(p.payment_num), 0)
          FROM k_payment p, Inserted i
          WHERE p.bill_num=i.bill_num
    UPDATE k_payment SET payment_num=@n+1
          WHERE bill_num=@bill and payment_num=0
```

Триггеры также удобно использовать для поддержания *ссылочной целостности*. Мы уже использовали *декларативную ссылочную целостность* с помощью *внешних ключей,* но она имеет исключительно запретительный характер. На самом же деле *политика* ссылочной целостности может быть пяти видов:

- IGNOGE – игнорировать,
- RESTRICT – запрещать,
- CASCADE – обрабатывать каскадным образом,
- SET DEFAULT – назначать значения по умолчанию,
- SET NULL – назначать NULL-значения.

Политика IGNORE означает, что мы не предусматриваем никаких проверок и ограничений.

Политика RESTRICT действует, когда мы применяем ограничения внешних ключей.

При использовании политики CASCADE мы должны предусмотреть собственную программную обработку, т.е. при изменении родительских таблиц вносить изменения в дочерние таблицы программным образом.

Политика SET DEFAULT состоит в том, что при изменении данных в родительских таблицах дочерним таблицам назначаются значения по умолчанию. Например, при удалении отдела мы можем записать его сотрудников в некоторый другой отдел, который мы считаем отделом по умолчанию.

Политика SET NULL похожа на предыдущую политику, только вместо значений по умолчанию мы назначаем `NULL`-значения.

Рассмотрим следующий пример. Пусть при удалении счета мы хотим удалять все строки его протокола. Пока у нас на этот случай действует внешний ключ, который запрещает удалять счет, для которого есть протокол.

Уберем этот внешний ключ:

```
ALTER TABLE k_protokol DROP CONSTRAINT fk_protokol_bill_num
```

Создадим триггер:

```
CREATE TRIGGER del_bill
    ON k_bill FOR DELETE
    AS
    DELETE FROM k_protokol WHERE bill_num IN
          (SELECT bill_num FROM Deleted d)
```

Протестируем триггер. Распечатаем сначала протокол счета с номером 5.

```
SELECT * FROM k_protokol WHERE bill_num=5

price_num bill_num kolvo    price_sum
-------------------------------------
1         5        1        1000.00
2         5        10       100.00

(2 row(s) affected)
```

Теперь удалим этот счет.

```
DELETE FROM k_bill WHERE bill_num=5
```

Снова распечатаем протокол этого счета.

```
SELECT * FROM k_protokol WHERE bill_num=5

price_num bill_num kolvo    price_sum
-------------------------------------

(0 row(s) affected)
```

Как видим, строки протокола тоже удалены.

Заметим, что этот триггер удаляет строки из таблицы `k_protokol`, вызывая тем самым ее собственный триггер. Такие цепочки вызовов триггеров могут быть и более длинными, главное – чтобы триггеры не конфликтовали друг с другом и не зацикливались.

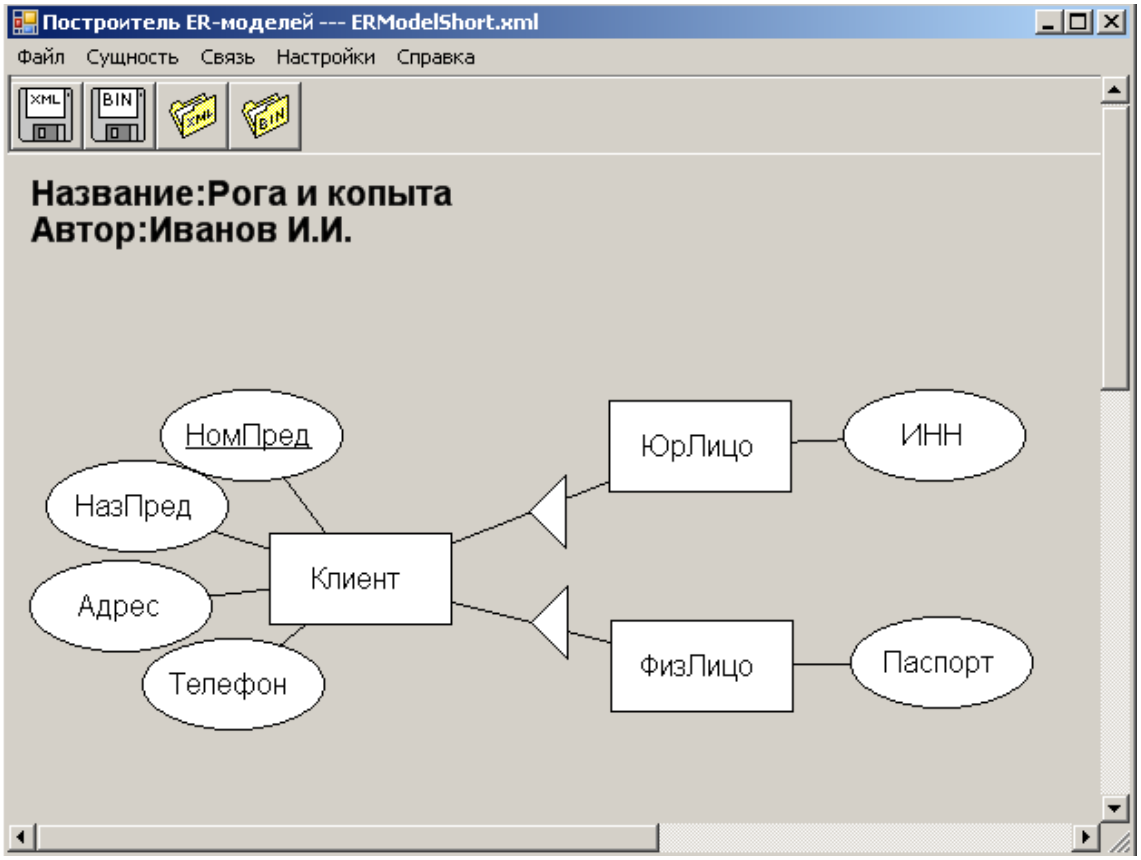
Задание для индивидуальной работы 9

Создайте несколько (не менее 2) триггеров для вашей базы данных. Например, с помощью триггеров вы можете реализовать каскадную политику ссылочной целостности.

Приложение 1. Работа с ERModeler

Программа ERModeler разработана специально для создания ER-моделей. Загрузить программу можно по адресу:
http://kek.ksu.ru/EOS/BD/ERModeler.zip .

После загрузки следует распаковать программу в произвольный каталог. Файл для запуска – **ERModel.exe**. Программа представляет собой исполняемый файл .NET, поэтому для функционирования программы необходимо, чтобы на компьютере был установлен .NET Framework версии 2.0 или выше.



Краткое описание функций программы содержится в справочной системе, в секции **«Часто задаваемые вопросы»**:

Как создать сущность?

Для создания сущности щелкните левой кнопкой мыши в любом месте рабочего поля. Эту настройку можно отключить через меню "Настройки".

Как изменить имя сущности?

Для того чтобы изменить имя сущности, выполните двойной щелчок мышью по этой сущности. Появится текстовое поле, в котором можно изменить имя сущности. Для подтверждения нового имени нажмите "Enter". Обратите внимание, что имя сущности не может быть длиннее 10 букв.

Можно ли перемещать сущность по рабочему полю?

Как сущности, так и атрибуты, и связи можно перетаскивать по рабочему полю. Для этого следует щелкнуть по объекту левой кнопкой мыши и, не отпуская кнопки, переместить объект в другое место. Обратите внимание, что сущность перемещается вместе со своими атрибутами.

Как создать атрибут сущности?

Для создания атрибута сущности сначала выделите сущность, т.е., щелкните на ней левой кнопкой мыши. Выделенная сущность будет отмечена черными квадратиками по углам. Теперь выберите пункт меню "Сущность" – "Новый атрибут". Атрибут будет добавлен над сущностью. Его можно передвинуть в любое место.

Как изменить имя атрибута?

Для того чтобы изменить имя атрибута, выполните двойной щелчок мышью по этому атрибуту. Появится текстовое поле, в котором можно изменить имя атрибута. Для подтверждения нового имени нажмите "Enter". Обратите внимание, что имя атрибута не может быть длиннее 10 букв. Другой способ изменить имя атрибута – использование диалогового окна "Свойства атрибута". Оно вызывается по щелчку **правой кнопкой** мыши на атрибуте.

Как изменить тип и длину атрибута?

Для того чтобы изменить тип и длину атрибута, используйте диалоговое окно "Свойства атрибута". Оно вызывается по щелчку **правой кнопкой** мыши на атрибуте.

Как создать ключевое поле?

Для того чтобы создать ключевое поле, используйте диалоговое окно "Свойства атрибута". Оно вызывается по щелчку **правой кнопкой** мыши на атрибуте. Название ключевого поля в модели будет подчеркнуто.

Как создать связь между сущностями?

Для создания связи между сущностями используйте диалоговое окно "Свойства связи". Оно вызывается через пункт меню "Связь" – "Новая связь". В этом окне выбирайте сущности и тип связи между ними. Можно создавать только бинарные связи.

Как изменить тип связи?

Для изменения типа связи используйте то же самое диалоговое окно "Свойства связи". Его можно вызвать, щелкнув **правой кнопкой** мыши на связи.

Как изменить название связи?

Для изменения названия связи используйте то же самое диалоговое окно "Свойства связи". Его можно вызвать, щелкнув **правой кнопкой** мыши на связи.

Как создать атрибут связи?

Для создания атрибута связи сначала выделите связь, т.е., щелкните на ней левой кнопкой мыши. Выделенная связь будет отмечена черными квадратиками по углам. Теперь выберите пункт меню "Связь" – "Новый атрибут". Атрибут будет добавлен над связью. Его можно передвинуть в любое место.

Как удалить сущность?

Для удаления сущности сначала выберите ее, а затем используйте пункт меню "Сущность" – "Удалить сущность".

Как удалить атрибут?

Для удаления атрибута сущности или связи сначала выберите атрибут, а затем используйте пункт меню "Сущность" – "Удалить атрибут" или "Связь" – "Удалить атрибут".

Как удалить связь?

Для удаления связи сначала выберите ее, а затем используйте пункт меню "Связь" – "Удалить связь".

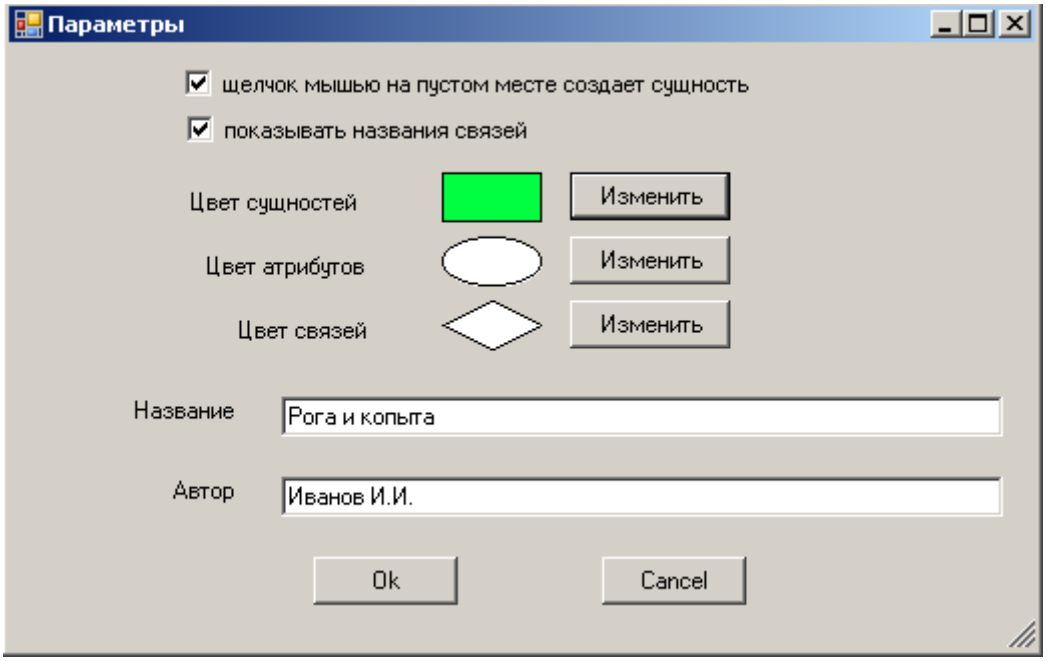
Как сохранить ER-модель для дальнейшей работы?

В пункте меню "Сохранить как..." можно использовать два формата – XML и двоичный файл. Преимущество файла XML в том, что его можно прочитать в текстовом редакторе или браузере. Преимущество двоичного формата в том, что получаются более короткие файлы.

Как преобразовать ER-модель в графический файл?

В пункте меню "Сохранить как..." есть возможность сохранить модель в виде PNG-файла.

Заметим также, что в окне настроек данной программы можно управлять некоторыми параметрами интерфейса ER-модели:



Обратите внимание, что цвета сущностей, атрибутов и связей сохраняются не в файле модели, а в реестре Windows (т.е., они связаны с определенным компьютером).

Приложение 2. Некоторые типичные ошибки SQL

При отладке программ неизбежно обнаруживаются разнообразные ошибки. Рассмотрим некоторые типичные ситуации при работе в **SQL Management Studio**.

Команда:

```
SELCT * FROM k_bill
```

Ошибка:

```
Msg 102, Level 15, State 1, Line 1
Неправильный синтаксис около конструкции "*".
```

Объяснение:

Синтаксическая ошибка, пропущена буква в слове `SELECT`.

Команда:

```
INSERT INTO k_firm (firm_name, firm_addr)
     VALUES(10, 'Сигма', 'Киев');
```

Ошибка:

```
Msg 110, Level 15, State 1, Line 1
Число столбцов в инструкции INSERT меньше числа значений, указанных в предложении VALUES. Число значений в предложении VALUES должно соответствовать числу столбцов, указанному в инструкции INSERT.
```

Объяснение:

В команде вставки в списке полей перечислены два поля, а в списке значений – три значения.

Команда:

```
INSERT INTO k_firm (firm_num, firm_name, firm_addr)
     VALUES(10, 'Сигма', 'Киев');
```

Ошибка:

```
Msg 544, Level 16, State 1, Line 1
Невозможно вставить явное значение для столбца идентификаторов в таблице "k_firm", когда параметр IDENTITY_INSERT имеет значение OFF.
```

Объяснение:

По умолчанию нельзя указывать явное значение для поля, у которого установлено свойство `IDENTITY`, т.е., для поля `firm_num`. Если

требуется явно указывать значения для таких полей, следует предварительно выполнить команду:

```
SET IDENTITY_INSERT ON
```

Команда:

```
INSERT INTO k_staff
     (staff_name, dept_num, staff_hiredate, staff_post)
     VALUES('Смит', 4, GETDATE(), 'Менеджер');
```

Ошибка:

```
Msg 547, Level 16, State 0, Line 1
Конфликт инструкции INSERT с ограничением FOREIGN KEY
"fk_staff_dept_num". Конфликт произошел в базе данных
"kontora", таблица "dbo.k_dept", column 'dept_num'.
Выполнение данной инструкции было прервано.
```

Объяснение:

Нарушено ограничение **внешнего ключа**: мы пытаемся вставить ссылку на несуществующий отдел с номером 4.

Команда:

```
DELETE FROM k_contract WHERE contract_num=1
```

Ошибка:

```
Msg 547, Level 16, State 0, Line 1
Конфликт инструкции DELETE с ограничением REFERENCE
"fk_bill_contract_num". Конфликт произошел в базе данных
"kontora", таблица "dbo.k_bill", column 'contract_num'.
Выполнение данной инструкции было прервано.
```

Объяснение:

Нарушено ограничение **внешнего ключа:** мы пытаемся удалить договор с номером 1, а к этому договору привязаны счета в таблице `k_bill`.

Команда:

```
SELECT  * FROM  k_contract
WHERE contract_date BETWEEN '01/03/2012' AND '31/03/2012'
```

Ошибка:

```
Msg 242, Level 16, State 3, Line 1
Преобразование типа данных char в тип данных datetime привело к
значению datetime за пределами диапазона.
```

Объяснение:

Видимо, на компьютере установлен другой формат даты.
Если вы хотите задать определенный формат даты, например, **день:месяц:год**, выполните команду:

```
SET DATEFORMAT dmy
```

Команда:

```
SELECT price_name, MIN(price_sum) FROM k_price
```

Ошибка:

```
Msg 8120, Level 16, State 1, Line 1
Столбец "k_price.price_name" недопустим в списке выбора,
поскольку он не содержится ни в статистической функции, ни в
предложении GROUP BY.
```

Объяснение:

Если используются **агрегирующие функции** без группировки, в списке полей могут присутствовать только агрегирующие функции.

Команда:

```
SELECT contract_num, contract_date, bill_num, bill_date
FROM k_bill, k_contract
WHERE  k_bill.contract_num=k_contract.contract_num
```

Ошибка:

```
Msg 209, Level 16, State 1, Line 1
Неоднозначное имя столбца "contract_num".
```

Объяснение:

Если в нескольких таблицах, используемых в запросе, есть поля с **одинаковыми** названиями, то для обращения к таким полям следует использовать синтаксис `имя_таблицы.имя_поля` или `псевдоним.имя_поля`.

Команда:

```
SELECT contract_num, contract_date FROM k_contract
     WHERE contract_num =
          (SELECT contract_num FROM k_bill
          WHERE bill_date
          BETWEEN '01/01/2012' AND '12/31/2012'
          AND k_contract.contract_num=k_bill.contract_num)
```

Ошибка:

```
Msg 512, Level 16, State 1, Line 1
Вложенный запрос вернул больше одного значения. Это запрещено,
когда вложенный запрос следует после =, !=, <, <=, >, >= или
используется в качестве выражения.
```

Объяснение:

Нельзя использовать обычные операции сравнения с подзапросом, если подзапрос возвращает *несколько* строк. Следует использовать ключевые слова `ALL` или `ANY`.

Команда:

```
INSERT INTO k_dept (dept_short_name, dept_full_name)
VALUES('Служба безопасности', 'Отдел №1');
```

Ошибка:

```
Msg 8152, Level 16, State 14, Line 1
Символьные или двоичные данные могут быть усечены.
Выполнение данной инструкции было прервано.
```

Объяснение:

Длина значения в строковом поле не должна превышать длину поля, заданную при создании таблицы.

Приложение 3. Реляционная алгебра и SQL

Рассмотрим, как связаны операции реляционной алгебры и язык SQL, т.е. приведем примеры запросов SQL, аналогичных операциям реляционной алгебры. В качестве примера базы данных будем использовать «Музыкантов».

Операция проекции **proj** выражается через SELECT с ключевым словом DISTINCT.

Получить все названия ансамблей:

```
proj НазАнс (Ансамбли)

SELECT DISTINCT НазАнс FROM Ансамбли
```

Операция выбора **sel** выражается через SELECT с ключевым словом WHERE.

Получить данные об ансамблях из России:

```
sel СтрАнс='Россия' (Ансамбли)

SELECT * FROM Ансамбли WHERE СтрАнс='Россия'
```

Условия также могут быть и сложными.

Получить имена музыкантов, родившихся в 20-м веке

```
SELECT ИмяМуз FROM Музыканты WHERE ДатаРожд>'31.12.1900' AND
ДатаРожд<'01.01.2001'
```

Операция соединения таблиц **join** может быть выражена несколькими способами.

Получить имена композиторов:

```
proj ИмяМуз (Музыканты join Сочинения)
```

Можно использовать связь таблиц через условие `WHERE`:

```
SELECT DISTINCT ИмяМуз FROM Музыканты M, Сочинения C WHERE
C.НомМуз=M.НомМуз
```

Можно использовать более современный синтаксис `JOIN ... ON`

```
SELECT DISTINCT ИмяМуз FROM Музыканты M JOIN Сочинения C ON
C.НомМуз=M.НомМуз
```

Если требуется вывести данные из одной таблицы, а условие накладывать на другую таблицу, то удобно использовать подзапросы, связанные и несвязанные.

```
SELECT DISTINCT ИмяМуз FROM Музыканты WHERE НомМуз IN
       (SELECT НомМуз FROM Сочинения)
```

или

```
SELECT DISTINCT ИмяМуз FROM Музыканты WHERE НомМуз = Any
       (SELECT НомМуз FROM Сочинения)
```

или

```
SELECT DISTINCT ИмяМуз FROM Музыканты M WHERE EXISTS
       (SELECT * FROM Сочинения C WHERE
             C.НомМуз=M.НомМуз)
```

Приведем пример сложного запроса, использующего данные из всех 6 таблиц базы данных.

Получить названия ансамблей, которые играли Моцарта на саксофоне:

```
proj  НазАнс
     (proj НомСоч (sel ИмяМуз='Моцарт' (Музыканты) join
               Сочинения)
       join
       proj НомСоч, НомАнс
          (proj НомИсп
          (sel Инструмент ='Саксофон' (Исполнители))
               join УчАнс join Исполнения)
       join Ансамбли )

SELECT НазАнс FROM Ансамбли WHERE НомАнс IN
(
```

```
        SELECT И1.НомАнс
        FROM Исполнения И1, Исполнители И2, Музыканты М,
               Сочинения С, УчАнс У
        WHERE И1.НомСоч=С.НомСоч AND С.НомМуз=М.НомМуз AND
               И1.НомАнс=У.НомАнс AND И2.НомИсп=У.НомИсп AND
               М.ИмяМуз='Моцарт' AND
               И2.Инструмент='Саксофон'
)
```

Операция объединения **union** соответствует нескольким командам SELECT, связанным ключевым словом UNION.

Получить общий список фамилий композиторов и дирижеров:

```
proj ИмяМуз (Музыканты join Сочинения)
     union
proj ИмяМуз (Музыканты join Исполнения)

SELECT DISTINCT ИмяМуз FROM Музыканты М, Сочинения С WHERE
С.НомМуз=М.НомМуз
UNION
SELECT DISTINCT ИмяМуз FROM Музыканты М, Исполнения И WHERE
И.НомМуз=М.НомМуз
```

Операция пересечения **intersection** может быть выражена несколькими способами.

Получить имена музыкантов, которые играют и на саксофоне, и на кларнете:

```
proj ИмяМуз (Музыканты join sel
          Инструмент='Саксофон'(Исполнители))
     intersection
proj ИмяМуз (Музыканты join sel
          Инструмент='Кларнет'(Исполнители))

SELECT DISTINCT ИмяМуз FROM Музыканты М1,
       Исполнители И1, Исполнители И2
WHERE М1.НомМуз=И1.НомМуз AND
       И1.Инструмент='Саксофон' AND
       И2.Инструмент='Кларнет' AND
       И2.НомМуз=И1.НомМуз
```

или

```
SELECT DISTINCT ИмяМуз
       FROM Музыканты М1, Исполнители И1
WHERE М1.НомМуз=И1.НомМуз AND
       И1.Инструмент='Саксофон' AND
       М1.НомМуз IN
              (SELECT НомМуз FROM Исполнители И2
                     WHERE И2.Инструмент='Кларнет')
```

или

```
SELECT DISTINCT ИмяМуз
       FROM Музыканты М1, Исполнители И1
       WHERE М1.НомМуз=И1.НомМуз AND
              И1.Инструмент='Саксофон' AND
              М1.НомМуз =ANY
              (SELECT НомМуз FROM Исполнители И2
                     WHERE И2.Инструмент='Кларнет')
```

или

```
SELECT DISTINCT ИмяМуз
       FROM Музыканты М1, Исполнители И1
WHERE М1.НомМуз=И1.НомМуз AND
       И1.Инструмент='Саксофон' AND
       EXISTS
              (SELECT * FROM Исполнители И2
              WHERE И2.Инструмент='Кларнет'
              AND И2.НомМуз=И1.НомМуз)
```

Операция вычитания **difference** также может быть выражена несколькими способами.

Получить имена музыкантов, котоые играют на саксофоне, но не играют на кларнете:

```
proj ИмяМуз (Музыканты join sel
          Инструмент='Саксофон'(Исполнители))
     diffrence
proj ИмяМуз (Музыканты join sel
          Инструмент='Кларнет'(Исполнители))
```

```
SELECT DISTINCT ИмяМуз
       FROM Музыканты М1, Исполнители И1
WHERE М1.НомМуз=И1.НомМуз AND
       И1.Инструмент='Саксофон' AND
       М1.НомМуз NOT IN
              (SELECT НомМуз FROM Исполнители И2
```

```
                WHERE И2.Инструмент='Кларнет')
```

или

```
SELECT DISTINCT ИмяМуз
       FROM Музыканты М1, Исполнители И1
       WHERE М1.НомМуз=И1.НомМуз AND
              И1.Инструмент='Саксофон' AND
              М1.НомМуз !=ALL
              (SELECT НомМуз FROM Исполнители И2
              WHERE И2.Инструмент='Кларнет')
```

или

```
SELECT DISTINCT ИмяМуз
       FROM  Музыканты М1, Исполнители И1
       WHERE М1.НомМуз=И1.НомМуз AND
              И1.Инструмент='Саксофон' AND
 NOT EXISTS
              (SELECT * FROM Исполнители И2
              WHERE И2.Инструмент='Кларнет'
              AND И2.НомМуз=И1.НомМуз)
```

Операция умножения **product** получается, если мы выполняем выборку из 2 таблиц, но не указываем условия связи.

Получить всевозможные пары имен музыкантов:

```
Музыканты2 aliases Музыканты

proj Музыканты.ИмяМуз, Музыканты2.ИмяМуз
     (Музыканты product Музыканты2)

SELECT М1.ИмяМуз, М2.ИмяМуз
FROM Музыканты М1, Музыканты М2
```

Очень интересно выглядит операция деления **division**. Она представляет собой двойное отрицание существования.

*Получить названия ансамблей, которые играли **все** произведения Моцарта (т.е., нет **ни одного** произведения Моцарта, которого они бы **не** играли):*

```
proj НазАнс
     (proj НомАнс, НомСоч (Исполнения)
```

```
                division
      proj НомСоч (sel ИмяМуз='Моцарт' (Музыканты)
                join Сочинения)
           join Ансамбли)

SELECT НазАнс FROM Ансамбли A WHERE NOT EXISTS
(
       SELECT * FROM Сочинения C, Музыканты M
       WHERE C.НомМуз=M.НомМуз AND ИмяМуз='Моцарт'
       AND NOT EXISTS
       (
              SELECT * FROM Исполнения И
              WHERE И.НомСоч=C.НомСоч AND
              И.НомАнс=A.НомАнс
       )
)
```

Литература

1. Грабер М. SQL. – М.: Лори. – 2007. – 672 с.
2. Виейра Р. Программирование баз данных Microsoft SQL Server 2005 для профессионалов. М.: Вильямс, Диалектика. – 2008. – 832с.
3. Ульман Дж. Базы данных на Паскале. – М.: Машиностроение. – 1990. – 386 с.
4. Крёнке Д. Теория и практика построения баз данных, 9-е издание. – СПб.: Питер. – 2005. – 900 с.
5. Учебники по SQL Server на сайте Microsoft: http://msdn.microsoft.com/ru-ru/library/ms167593.aspx