Министерство образования и науки Российской Федерации
Федеральное агентство по образованию

Санкт-Петербургский государственный университет информационных технологий, механики и оптики

А.А. Бобцов, В.В. Шиегин

Банки и базы данных. Основы работы с MS Access. Часть 2 (для разработчиков)

Учебное пособие

Санкт-Петербург

2005

УДК 681.3

Бобцов А.А., Шиегин В.В. Банки и базы данных. Основы работы с MS Access. Часть 2 (для разработчиков). Учебное пособие. – СПб., 2005. - 57 с.

Рецензенты: Л.С. Лисицына, к.т.н., доцент, зав. каф. КОТ СПбГУ ИТМО
А.В. Лямин, к.т.н., доцент, директор ЦДО СПбГУ ИТМО

Учебное пособие предназначено для дисциплины ОПД.Ф.14 “Банки и базы данных” для специальности 230202 – «Информационные технологии в образовании», а также для курсов повышения квалификации работников образования по программам соответствующего содержания.
Учебное пособие является продолжением Части 1 (для пользователей) и дает читателю знания, необходимые при проектировании и реализации баз данных на основе офисного пакета MS Access.
Методы проектирования и анализа моделей баз данных, рассмотренные в пособии, применимы в любой реляционной системе управления базами данных.

Печатается по решению УМС факультета ИТиП СПбГУ ИТМО

Содержание

Введение

Данное учебное пособие знакомит с основами проектирования и реализации баз данных на основе офисного пакета MS Access. Пособие ориентировано на обучаемых, имеющих опыт программирования и может служить методической поддержкой практических занятий. За время обучения будут рассмотрены следующие темы:

- назначение и возможности языка структурированных запросов SQL;
- применение языка SQL при создании запросов и других объектов баз данных;
- синтаксис SQL-операторов в системе управления базами данных (СУБД) MS Access;
- типы данных, функции и операции, применяемые в MS Access;
- проектирование реляционных баз данных (построение модели, задача нормализации).

Данное пособие является продолжением пособия [1]. Предполагается, что читатель знаком с изложенным там материалом. Другие источники, перечисленные в списке литературы, также могут быть полезны для более подробного изучения вопросов проектирования и реализации баз данных.

1. Язык SQL

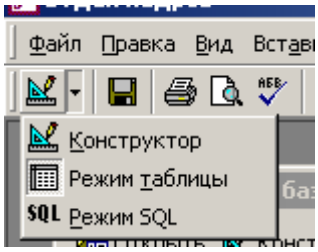
Аббревиатура SQL означает Structured Query Language (структурированный язык запросов). Синтаксис SQL разрабатывался для удобства формирования запросов «близко к естественному английскому». Предполагалось, что его смогут использовать рядовые пользователи баз данных.

В настоящее время пользователи обычно общаются с БД через более удобный интерфейс клиентских приложений, но взаимодействие между приложением и СУБД часто происходит с использованием SQL. Составленный запрос на языке SQL передается клиентом серверу, на сервере производится его выполнение, после чего клиенту передается сформированный результирующий набор. В системах «клиент-сервер» SQL чрезвычайно удобен, поскольку позволяет выполнить достаточно сложную обработку на сервере, без передачи промежуточных данных клиенту.

Существуют несколько стандартов языка [6] (различающихся временем их опубликования), но его реализация в различных СУБД может не полностью соответствовать этим стандартам.

В Access при обращении к БД также применяется язык SQL (Access изнутри организован по системе «клиент-сервер»). Любой запрос, построенный с помощью мастера или конструктора, имеет соответствующее представление на языке SQL. Конструктор – лишь визуальное средство для создания запросов. В Access имеется возможность редактировать запросы непосредственно в режиме SQL. Причем, не всякий составленный на SQL запрос, может быть отображен в режиме конструктора – SQL имеет более широкие возможности, чем визуальный конструктор.

Для переключения режимов отображения запросов используется кнопка «**Вид**» панели инструментов.



На рис. 1.1 показано, как выглядит один и тот же запрос в трех различных режимах – конструктора, таблицы и SQL. При освоении языка SQL бывает полезно составить запрос с помощью конструктора, а затем просмотреть или модифицировать его в режиме SQL.

Запрос на языке SQL может быть напрямую задан в свойствах «**Источник записей**» или «**Источник строк**» элементов форм и отчетов. Это позволяет обойтись без создания отдельного объекта «Запрос».

Кроме того, SQL используется при работе с внешними источниками данных (например, СУБД Oracle или Microsoft SQL Server). Однако, в этом случае используется уже синтаксис SQL этих СУБД, а не «внутренний» SQL.

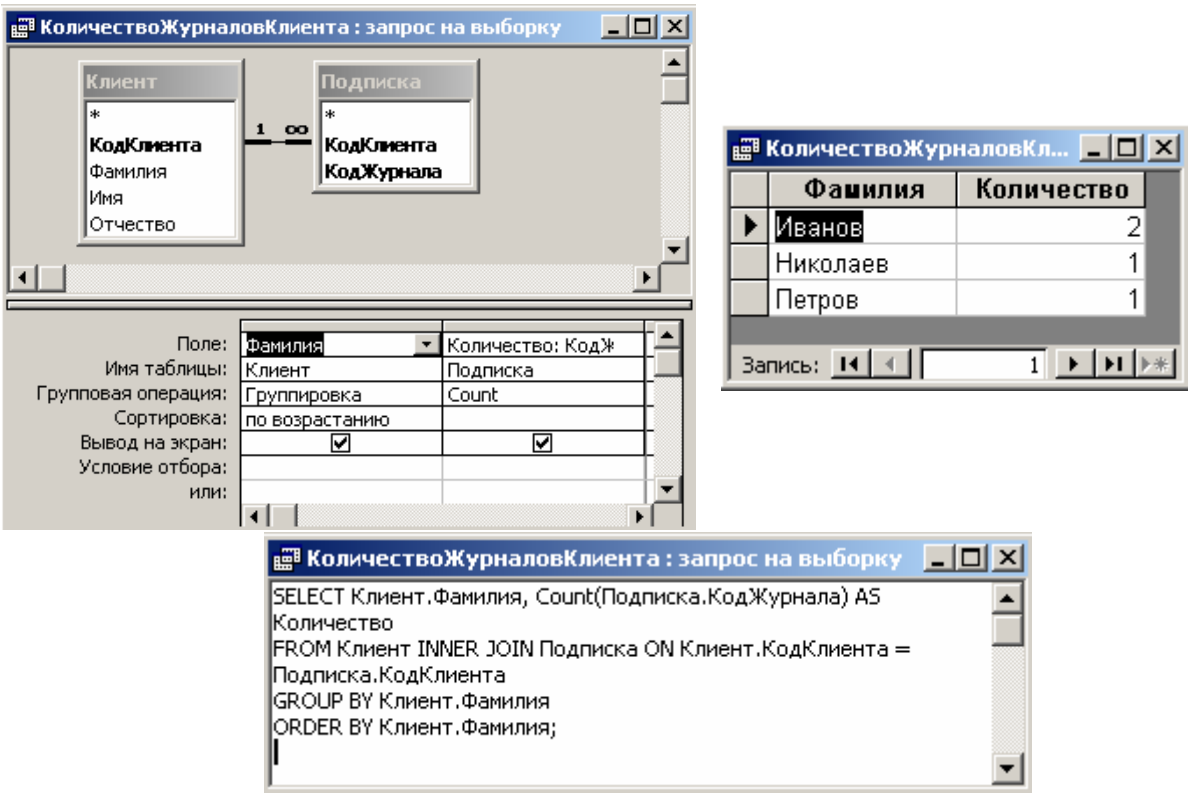


Рис. 1.1. Режимы отображения запроса

1.1. SQL в простых запросах на извлечение данных

Чаще всего возникает задача построения запросов на извлечение данных. Для этих целей используется SQL-оператор SELECT. С него мы и начнем изучение языка SQL.

Простейшая форма оператора SELECT

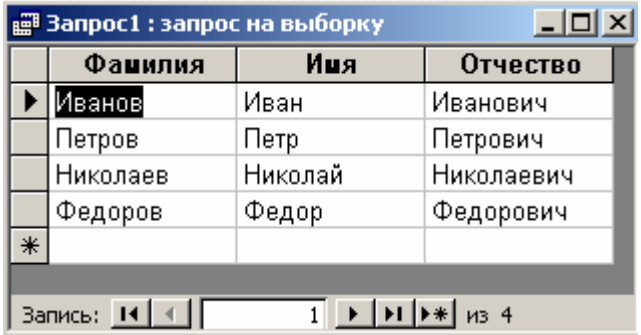
SELECT <список полей>
FROM <список источников>**;**

Список полей – имена полей (столбцов), которые следует извлечь из источников (таблиц) и поместить в результирующий набор.

Пример 1.1.

SELECT Фамилия, Имя, Отчество
FROM Клиент**;**

Результат работы этого запроса:



Смысл этого запроса таков: извлечь поля «Фамилия», «Имя», «Отчество» из таблицы «Клиент».

Пример 1.2.

SELECT *
FROM Клиент**;**

Результат работы запроса:

Запрос1 : запрос на выборку

	КодКлиента	Фамилия	Имя	Отчество
▶	1	Иванов	Иван	Иванович
	2	Петров	Петр	Петрович
	3	Николаев	Николай	Николаевич
	4	Федоров	Федор	Федорович
*	(Счетчик)			

Запись: 1 из 4

Звездочка («*») означает «все поля» - если нужно извлечь не одно или несколько конкретных полей из таблицы, а все имеющиеся поля, не требуется перечислять их поименно.

Если говорить более точно, то простейшая форма оператора SELECT может не содержать даже предложения FROM.

Пример 1.3.

SELECT 123, “qwerty”, Date();

Результатом этого запроса будет строка, содержащая 3 столбца – число, текстовое значение и текущую дату (в предложении SELECT указываются не имена столбцов, а константы или выражения, возвращающие конкретные значения). Здесь нет предложения FROM, поскольку для получения данных не используются никакие источники (таблицы или запросы).

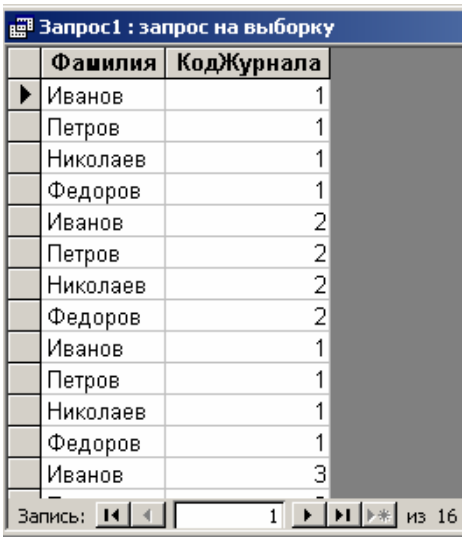
Использование нескольких источников (таблиц).

Пример 1.4. Есть две таблицы

Построим запрос, выводящий фамилии клиентов и коды журналов.

SELECT Клиент.Фамилия, Подписка.КодЖурнала
FROM Клиент, Подписка**;**

Результат представляет собой **декартово произведение** исходных таблиц. Каждая запись таблицы «**Клиент**» объединяется с каждой записью таблицы «**Подписка**»

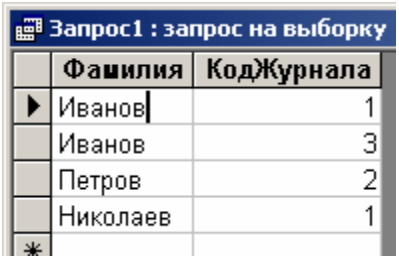


Следует ограничить результирующее множество комбинациями только тех записей этих двух таблиц, которые связаны между собой (содержат одинаковые значения в полях «**КодКлиента**»).

В предложении «**FROM**» вместо оператора объединения «**,**» (запятая) будем использовать оператор «**INNER JOIN**» - внутреннее объединение. Оператор «**INNER JOIN**» позволяет наложить ограничение на объединяемые записи. Предложение **ON <условие>** определяет связь между полями объединяемых таблиц.

SELECT Клиент.Фамилия, Подписка.КодЖурнала
FROM Клиент **INNER JOIN** Подписка
ON Клиент.КодКлиента = Подписка.КодКлиента;

Результат содержит только комбинации записей, для которых выполняется заданное условие.



Рассмотрим подробнее разные виды объединения.

- **INNER JOIN** (внутреннее объединение). В результат включаются только те записи из обеих таблиц, которые связаны между собой.
- **LEFT JOIN** (левое внешнее объединение). В результат включаются все записи из первой таблицы. Если для них нет связанных записей во второй таблице, соответствующие поля результата будут пустыми.
- **RIGHT JOIN** (правое внешнее объединение). Операция, зеркально симметричная левому объединению. Включаются все записи из второй таблицы и связанные с ними записи из первой.

В режиме конструктора тип объединения можно изменить заданием свойств связи. Эта возможность была рассмотрена в [1], при описании работы с конструктором запросов. При создании запроса в режиме конструктора для таблиц,

связи которых заданы в схеме данных, по умолчанию автоматически определяется операция «**INNER JOIN**».

Пример 1.5. Заменим в созданном запросе (пример 1.4) тип объединения на «**LEFT JOIN**».

SELECT Клиент.Фамилия, Подписка.КодЖурнала
FROM Клиент **LEFT JOIN** Подписка
ON Клиент.КодКлиента = Подписка.КодКлиента;

Результат работы измененного запроса:

Запрос1 : запрос на выборку

	Фамилия	КодЖурнала
▶	Иванов	1
	Иванов	3
	Петров	2
	Николаев	1
	Федоров	
*		

Здесь присутствует фамилия «**Федоров**», для которой нет соответствий в таблице «Журнал». Вторая колонка этой строки содержит значение «**Null**» (пусто).

Группировка, сортировка, имена столбцов. Продолжим доработку составленного запроса. Заметим, что фамилия «Иванов» в результатах запроса повторяется, поскольку для этого клиента оформлена подписка на два журнала.

Пример 1.6. Изменим запрос таким образом, чтобы фамилия каждого клиента выводилась только один раз, и для него отображалось общее количество выписанных журналов, а не коды каждого из журналов.

Зададим группировку записей по фамилии (фамилии не должны повторяться) и для групп определим групповую операцию «подсчет количества».

SELECT Клиент.Фамилия, **Count**(Подписка.КодЖурнала)
FROM Клиент **LEFT JOIN** Подписка
ON Клиент.КодКлиента = Подписка.КодКлиента
GROUP BY Клиент.Фамилия;

Здесь подчеркнуты фрагменты, которые отличают этот запрос от предыдущего.

В предложении **GROUP BY** могут быть перечислены несколько полей через запятую. Если группировка производится по нескольким полям, объединяться в группу будут строки, для которых попарно равны значения всех группируемых полей. Для всех полей, перечисленных в предложении **SELECT**, но не вошедших в предложение **GROUP BY**, должны быть определены групповые операции.

Результат работы запроса:

Пример 1.7. Название второго столбца (в примере выше) сформировано автоматически. Надо задать осмысленное название. Кроме того, является лишь совпадением то, что фамилии расположены по алфавиту. Следует явно указать способ сортировки результирующего набора. Внесем соответствующие изменения в запрос.

SELECT Клиент.Фамилия, **Count**(Подписка.КодЖурнала) **AS** Количество
FROM Клиент **LEFT JOIN** Подписка
ON Клиент.КодКлиента = Подписка.КодКлиента
GROUP BY Клиент.Фамилия
ORDER BY Клиент.Фамилия **ASC**;

Результат работы запроса:

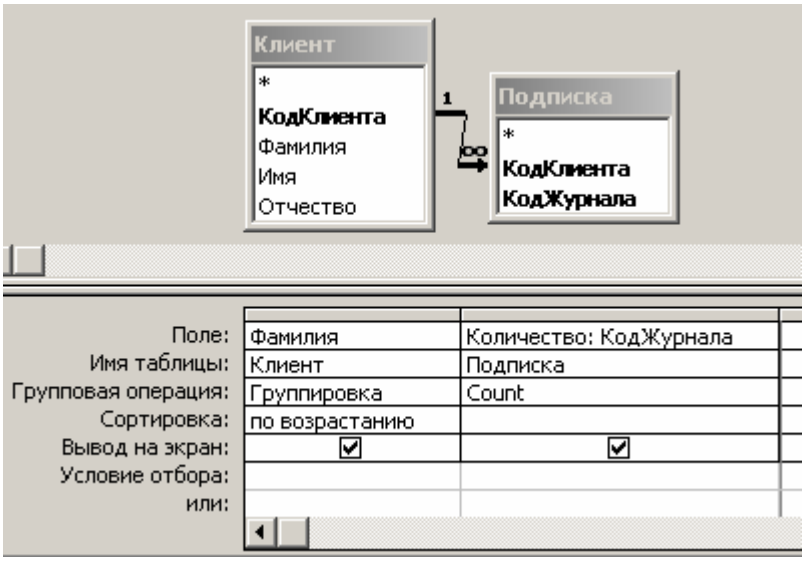
Ключевое слово **AS** указывает имя (псевдоним) для столбца. Если имя не задано явно, используется соответствующее ему имя поля исходной таблицы. Если же столбец формируется с помощью некоторого выражения, Access присваивает ему имя самостоятельно (как было в предыдущем примере).

Псевдонимы можно задавать не только для столбцов, но и для источников данных (таблиц), указанных в предложении **FROM**. Тогда к ним можно обращаться внутри запроса по новому имени.

Параметры сортировки задаются в предложении **ORDER BY**. В предложении можно перечислять несколько полей через запятую. Сортировка будет выполняться сначала по первому полю, затем по второму (если совпадают значения первого поля) и т.п.

В предложении **ORDER BY** для каждого из столбцов может указываться одно из ключевых слов, задающих направление сортировки – **ASC** (по возрастанию) или **DESC** (по убыванию). Если направление не указано, подразумевается значение **ASC**.

Созданный запрос с точки зрения Access достаточно тривиален. Его можно просмотреть в режиме конструктора (как уже говорилось, все запросы, созданные в режиме конструктора могут быть представлены в режиме SQL, но не все, созданные в режиме SQL, могут быть представлены в режиме конструктора).



Линия связи, соединяющая таблицы, помечена стрелкой. Это указывает на то, что используется внешнее объединение, а не внутреннее.

Ограничение результирующих наборов. Оператор **SELECT** извлекает данные из одной или нескольких таблиц и из всех возможных комбинаций оставляет только те, которые соответствуют заданным критериям отбора.

Можно считать, что при этом последовательно выполняются две основные операции – **умножение** и **сужение** [5, 9 – 11, 13]. **Умножение** – построение всех возможных комбинаций записей (**декартово произведение** таблиц). **Сужение** (его также называют выборкой, ограничением или селекцией) – отсечение «лишних» комбинаций. Умножение выполняется в предложении **FROM** и там же с помощью слова **ON** может быть выполнено предварительное сужение (из всех комбинаций записей остаются только связанные между собой).

Для выполнения дальнейшего сужения в операторе **SELECT** могут присутствовать еще два вида предложений: **WHERE** и **HAVING**.

Предложение **WHERE <условие>** располагается после предложения **FROM** и позволяет наложить дополнительные ограничения на результат объединения. Ограничения, заданные словом **ON** иногда могут быть перенесены также в предложение **WHERE**.

Предложение **HAVING <условие>** может располагаться после предложения **GROUP BY** и применяться к данным каждой сформированной группы. При использовании предложения **HAVING** без предложения **GROUP BY**, оно применяется ко всей результирующей таблице и действует аналогично предложению **WHERE**.

Пример 1.8. Дополним разработанный ранее запрос новыми ограничениями.

SELECT Клиент.Фамилия, **Count**(Подписка.КодЖурнала) **AS** Количество
FROM Клиент **LEFT JOIN** Подписка
ON Клиент.КодКлиента = Подписка.КодКлиента
WHERE Left(Клиент.Фамилия,1) = "И" **AND** Подписка.КодЖурнала > 1
GROUP BY Клиент.Фамилия

HAVING Count(Подписка.КодЖурнала) > 0
ORDER BY Клиент.Фамилия **ASC**;

В предварительный результат (после предложения **WHERE**) входят записи только для тех клиентов, фамилия которых начинается с «И» и при этом используются только журналы, коды которых больше «1». К сформированному набору применяется операция группировки. Уже после группировки исключаются те клиенты, у которых нет подписки (число выписанных журналов равно «0»), при подсчете количества не учитывается информация о подписке на журналы, исключенные ранее в предложении **WHERE**. То, что осталось, сортируется в алфавитном порядке.

Получившийся запрос не является оптимальным, он был составлен исключительно в демонстрационных целях. Например, способ объединения **LEFT JOIN** используется для включения фамилий клиентов, которые не имеют подписки. Поскольку такие клиенты все равно гарантированно не попадут в результирующий набор, вместо **LEFT JOIN** здесь вполне можно использовать **INNER JOIN**.

1.2. Подробнее о синтаксисе SQL

Общая форма оператора SELECT

SELECT [**DISTINCT**] список_выражений|*
[**INTO** новая_таблица]
[**FROM** объединение_источников]
[**WHERE** условие]
[**GROUP BY** список_столбцов]
[**HAVING** условие]
[**UNION** [**ALL**] **SELECT** …]
[**ORDER BY** список_столбцов]

Некоторые входящие в состав оператора предложения были рассмотрены ранее на простых примерах. Ниже будут показаны более сложные конструкции.

Необязательное ключевое слово **DISTINCT** отвечает за то, чтобы в результирующем наборе не было полностью совпадающих строк (записей). Повторяющиеся строки будут исключены.

Вложенные запросы в предложении FROM. В операторе **SELECT** в предложении **FROM** перечисляются имена таблиц или запросов, которые являются источниками данных для этого запроса.

Пример 1.9.

SELECT Клиент.Фамилия, Подписка.Журнал
FROM Клиент, Подписка
…

Таблицы постоянно хранят данные, а запросы формируют временные результирующие наборы в форме таблиц в соответствии с определенными критериями отбора.

С точки зрения использования в качестве источников данных при составлении запроса, нет разницы между запросами и «постоянными» таблицами. В языке SQL есть возможность не только обращаться по имени к созданным ранее объектам – запросам, но и определять «**вложенные**» («**подчиненные**») запросы непосредственно внутри основного запроса.

Пример 1.10 (один из источников данных – вложенный запрос).

```
SELECT Клиент.Фамилия, Подписка2.Журнал
FROM Клиент, (SELECT * FROM Подписка) AS Подписка2
…
```

Ключевое слово «**AS**» задает имя (псевдоним), по которому следует обращаться к полям временной таблицы, сформированной вложенным запросом, в пределах «главного» запроса.

Результаты работы запросов в первом и втором примерах идентичны. Вложенный запрос в данном случае не имеет практического смысла. Однако вложенные запросы могут иметь и более сложную структуру, и в некоторых ситуациях необходимы.

Вложенные запросы, в свою очередь, также могут содержать внутри себя вложенные запросы.

Пример 1.11 (трехуровневый запрос, действует аналогично двум предыдущим).

```
SELECT Клиент.Фамилия, Подписка2.Журнал
FROM Клиент, (SELECT * FROM
                  (SELECT * FROM Подписка)
              ) AS Подписка2
…
```

В операторе «**SELECT**», применяемом при создании вложенных запросов, могут быть использованы любые предложения, как и в операторах верхнего уровня («**FROM**», «**WHERE**», «**GROUP BY**», «**ORDER BY**» …).

Действующий пример с использованием вложенного запроса будет приведен ниже.

Вложенные запросы в предложении WHERE. Вложенные запросы могут быть использованы не только в качестве источников данных, но и в предложении **WHERE**, при определении ограничений результирующего набора.

Пример 1.12. Пусть существует таблица

Зарплата(<u>Номер</u>, ФИО, Оклад)

Запрос, выводящий ФИО и оклад работников, оклад которых выше среднего, может иметь следующий вид:

```
SELECT ФИО, Оклад
FROM Зарплата
WHERE Оклад >
    (SELECT AVG(Оклад) FROM Зарплата)
ORDER BY ФИО;
```

Для того чтобы выполнить проверку условия «выше среднего», требуется вычислить это самое среднее. Эту работу выполняет вложенный запрос, который обращается к той же самой таблице «Зарплата».

Вложенный запрос, для которого используются подобные операции сравнения, должен гарантированно выдавать только одну результирующую запись, состоящую только из одного столбца.

Пример 1.13. Запрос выводит список работников, получающих максимальный оклад:

```
SELECT ФИО, Оклад
FROM Зарплата
WHERE Оклад =
    (SELECT MAX(Оклад) FROM Зарплата)
ORDER BY ФИО;
```

Вместе с операциями сравнения могут быть использованы операторы **ALL** (все) и **ANY** (какой-нибудь). **ALL** требует, чтобы заданное условие выполнялось для всех записей из вложенного запроса, **ANY** - хотя бы для одной.

Пример 1.14. Запрос выполняет те же действия, что и предыдущий:

```
SELECT ФИО, Оклад
FROM Зарплата
WHERE Оклад >= ALL
    (SELECT Оклад FROM Зарплата)
ORDER BY ФИО;
```

При использовании операторов **ALL** и **ANY**, вложенный запрос может выдавать любое количество строк (записей), но в нем также должен быть только один столбец (запросы такого вида называют **скалярными**).

Пример 1.15. Вывести список работников, оклад которых НЕ САМЫЙ НИЗКИЙ (есть кто-то, кто получает еще меньше):

```
SELECT ФИО, Оклад
FROM Зарплата
WHERE Оклад > ANY
    (SELECT Оклад FROM Зарплата)
ORDER BY ФИО;
```

Пример 1.16. Эту же задачу можно решить иначе, без оператора **ANY**:

```
SELECT ФИО, Оклад
FROM Зарплата
WHERE Оклад <>
    (SELECT MIN(Оклад) FROM Зарплата)
ORDER BY ФИО;
```

Для работы с вложенными запросами также существуют операторы **IN** и **NOT IN**. Оператор **IN** требует, чтобы значение его левого аргумента содержалось в результирующем наборе правого аргумента (вложенного запроса). Оператор **NOT IN** имеет противоположное назначение.

Пример 1.17. Список работников с окладом выше среднего:

```
SELECT ФИО, Оклад
FROM Зарплата
WHERE Оклад NOT IN (
    SELECT Оклад FROM Зарплата
    WHERE Оклад <=
        (SELECT AVG(Оклад) FROM Зарплата)
    )
ORDER BY ФИО;
```

Данный запрос составлен исключительно с целью проиллюстрировать работу оператора **IN**, ранее (пример 1.12) был приведен более простой запрос, выполняющий те же действия.

Нетрудно убедиться, что **IN** можно заменить на «**= ANY**», а **NOT IN** на «**<> ALL**».

Как видно из приведенного примера, вложенные запросы в предложении **WHERE** также могут иметь больше одного уровня вложенности.

Для обработки вложенных запросов могут использоваться операторы **EXISTS** (существует) и **NOT EXISTS** (не существует). Оператор **EXISTS** возвращает значение «истина», если вложенный запрос содержит хотя бы одну запись (не важно, с какими данными). Оператор **NOT EXISTS** имеет противоположное назначение.

Пример 1.18. Список работников, у которых НЕ самый низкий оклад (это уже третий вариант запроса для решения той же самой задачи – см. примеры 1.15, 1.16):

```
SELECT ФИО, Оклад
FROM Зарплата AS Зп1
WHERE EXISTS
    (SELECT Оклад FROM Зарплата WHERE Оклад < Зп1.Оклад)
ORDER BY ФИО;
```

Для таблицы «Зарплата» из внешнего запроса определяется псевдоним «Зп1», поскольку имя ее поля используется во вложенном запросе, в котором фигурирует таблица с таким же именем и полями.

Для вложенных запросов, взаимодействующих с оператором **EXISTS**, нет ограничений по количеству полей, поскольку сами извлекаемые данные оператором никак не обрабатываются, его интересует лишь их принципиальное наличие или отсутствие.

Предложение UNION … SELECT. Иногда может возникнуть необходимость объединить данные из нескольких источников, как показано на рис. 1.2.

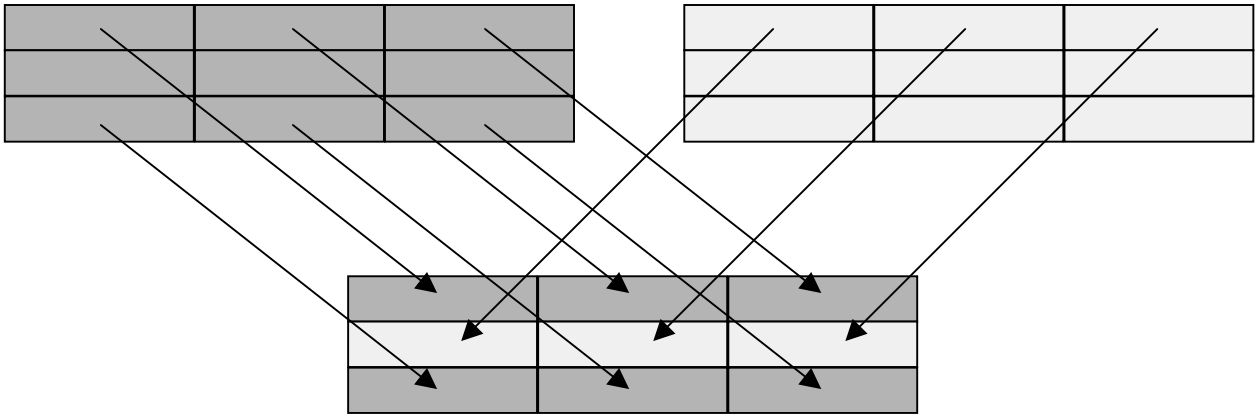


Рис. 1.2. Объединение данных из двух источников

Для этой цели используется предложение **UNION** оператора **SELECT** (см. выше общую форму оператора **SELECT**). В основном операторе **SELECT** выполняется выборка, формирующая первый результирующий набор, затем, после ключевого слова **UNION** помещается еще один оператор **SELECT**, формирующий второй набор. При выполнении такого запроса оба набора объединяются. Объединяться могут данные более чем двух источников. При этом предложение **UNION … SELECT** будет использовано несколько раз.

Количество и типы данных полей, формируемых каждым из запросов **SELECT**, должны быть одинаковыми. Имена (псевдонимы) столбцов можно задавать только в основном операторе **SELECT**. Если их задать в дополнительных операторах, Access их проигнорирует.

Пример 1.19. Есть две таблицы. Одна содержит данные о книгах, вторая – о журналах:

Книга : таблица

ИДКниги	НазваниеКниги	АвторКниги	ГодИзданияКниги
1	Война и Мир	Толстой Л.Н.	1981
2	Идиот	Достоевский Ф	1975

Журнал : таблица

ИДЖурнала	НазваниеЖурнала	НомерЖурнала	ГодЖурнала
1	Огонек	5	1995
2	Звезда	2	1984

Построим запрос, объединяющий данные из этих двух таблиц в результирующий набор, содержащий поля «**Заголовок**», «**Год**» и «**Тип**».

```
SELECT НазваниеЖурнала + " / № " + STR(НомерЖурнала)  AS Заголовок,
    ГодЖурнала AS Год,
    "Журнал" AS Тип
FROM Журнал
UNION SELECT НазваниеКниги + " / " + АвторКниги,
        ГодИзданияКниги,
        "Книга"
```

FROM Книга
ORDER BY Заголовок;

Результат работы запроса:

Журналы и книги : запрос на объединение

Заголовок	Год	Тип
Война и Мир / Толстой Л.Н.	1981	Книга
Звезда / № 2	1984	Журнал
Идиот / Достоевский Ф.М.	1975	Книга
Огонек / № 5	1995	Журнал

Предложение **ORDER BY** применяется не ко второму запросу, а к результату объединения.

По умолчанию при слиянии двух наборов, Access объединяет строки, которые полностью совпадают. Если этого делать не нужно, следует использовать ключевое слово **ALL** (**… UNION ALL SELECT …**).

Предложение INTO … Использование в операторе **SELECT** конструкции вида
SELECT <поля> **INTO** <новая таблица>
…
позволяет поместить результат запроса в таблицу, которая будет для этого создана.

Пример 1.20.
SELECT ИДЖурнала, НазваниеЖурнала **INTO** ЖурналНовая
FROM Журнал;

Создается таблица с названием «**ЖурналНовая**» и в нее помещаются следующие данные:

ЖурналНовая : таблица

ИДЖурнала	НазваниеЖурнала
1	Огонек
2	Звезда
(Счетчик)	

Конструкция TRANSFORM … PIVOT. Данная конструкция предназначена для создания перекрестных запросов (см. [1]). Общая форма:
TRANSFORM <выражение>
SELECT <предложение>
FROM <предложение>
…
GROUP BY <предложение>
PIVOT <имя поля>;

Конструкция **TRANSFORM … PIVOT** отсутствует в стандарте языка SQL.

Значения полей, перечисленных в предложении **SELECT**, образуют заголовки строк таблицы. Эти же поля должны быть указаны в предложении **GROUP BY**. Значения поля, указанного в предложении **PIVOT**, образуют заголовки столбцов таблицы. Значения выражения, записанного в предложении

TRANSFORM, размещаются в ячейках таблицы. Данное выражение должно содержать групповую операцию.

Пример 1.21.

```
TRANSFORM Count(Подписка.Журнал) AS Количество
SELECT Список.Фамилия
FROM
    (SELECT Клиент.КодКлиента, Клиент.Фамилия,
        Журнал.КодЖурнала, Журнал.Название
    FROM Журнал, Клиент
    ) AS Список
    LEFT JOIN Подписка
    ON (Список.КодЖурнала=Подписка.Журнал) AND
        (Список.КодКлиента=Подписка.Клиент)
GROUP BY Список.Фамилия
PIVOT Список.Название;
```

Результат работы запроса:

Фамилия	Журнал 1	Журнал 2	Журнал 3	Журнал 4
Иванов	0	0	1	0
Николаев	1	0	0	0
Петров	1	1	0	0
Федоров	0	0	0	0

Заголовки строк – фамилии клиентов, заголовки столбцов – названия журналов, а содержимое таблицы – количество подписок данного клиента на данный журнал.

Этот запрос отличается от представленного в [1] при описании конструктора запросов тем, что в нем отображены все клиенты, независимо от того, подписаны ли они на какой-то журнал, и все журналы, независимо от того, подписан ли на них кто-то из клиентов.

Для получения такого результата был использован вложенный запрос, формирующий декартово произведение таблиц «Клиент» и «Журнал» (т.е. все возможные их пары):

КодКлиента	Фамилия	КодЖурнала	Название
1	Иванов	1	Журнал 1
1	Иванов	2	Журнал 2
1	Иванов	3	Журнал 3
1	Иванов	4	Журнал 4
2	Петров	1	Журнал 1
2	Петров	2	Журнал 2
2	Петров	3	Журнал 3
2	Петров	4	Журнал 4
3	Николаев	1	Журнал 1
3	Николаев	2	Журнал 2
3	Николаев	3	Журнал 3
3	Николаев	4	Журнал 4
4	Федоров	1	Журнал 1
4	Федоров	2	Журнал 2
4	Федоров	3	Журнал 3
4	Федоров	4	Журнал 4

Запись: 1 из 16

Результат вложенного запроса с помощью **LEFT JOIN** объединяется с данными из таблицы «Подписка», после чего выполняется группировка строк (по фамилии и названию) и групповая операция Count().

Поскольку этот перекрестный запрос содержит вложенный запрос, его невозможно адекватно представить в режиме конструктора (вернее, конструктор способен отобразить такой запрос, но не дает возможности его создать или отредактировать). Изменять его можно только в режиме SQL.

Для того чтобы все-таки реализовать то же самое, не прибегая к SQL, нужно создать отдельный объект-запрос, выполняющий то же, что и вложенный запрос здесь, затем использовать его в качестве источника при построении нового запроса.

Оператор INSERT INTO. Оператор добавляет новые записи в таблицу.

INSERT INTO имя_таблицы
[(имя_столбца[,имя_столбца …])]
VALUES (значение[, значение …])

Добавляет в таблицу строку, присваивая указанным полям перечисленные значения.

INSERT INTO имя_таблицы
[(имя_столбца[,имя_столбца …])]
SELECT …

Добавляет в таблицу строки, сформированные оператором **SELECT**.

Пример 1.22.

INSERT INTO
Журнал (НазваниеЖурнала, НомерЖурнала, ГодЖурнала)
VALUES ("Новый мир", 4, 1987)**;**

Оператор DELETE FROM

DELETE FROM имя_таблицы
[**WHERE** условие]

Оператор удаляет из указанной таблицы записи в соответствии с заданным условием. При отсутствии предложения **WHERE** удаляются все записи.

Пример 1.23.

DELETE FROM Журнал
WHERE НазваниеЖурнала="Новый мир";

Оператор UPDATE

UPDATE имя_таблицы
SET
имя_столбца = значение|(**SELECT**…)
[,имя_столбца = значение|(**SELECT**…)…]
[**WHERE** условие]

Устанавливает значения для указанных столбцов в строках таблицы, соответствующих условию отбора. Значение может быть получено как результат вложенного запроса.

Пример 1.24 (увеличение минимального размера оплаты труда – МРОТ).

UPDATE Зарплата
SET Оклад = 1200
WHERE Оклад < 1200**;**

Оператор CREATE TABLE. Оператор создает таблицу с заданным именем и набором столбцов.

Упрощенная форма:

CREATE TABLE имя_таблицы (
имя_столбца **тип_данных**[**(размер)**]
[,имя_столбца **тип_данных** [**(размер)**] …]
)

Названия и описание основных типов данных, допустимых при создании таблиц, приведены ниже.

Пример 1.25. Создается таблица с именем «МояТаблица», содержащая три столбца (поля) с именами «Поле1», «Поле2» и «Поле3».

CREATE TABLE МояТаблица (
Поле1 **COUNTER**,
Поле2 **INTEGER**,
Поле3 **CHAR**
);

Кроме имен и типов данных, при создании таблицы могут быть заданы различные ограничения. Различают **простые** и **составные** ограничения. Простые ограничения задаются в форме атрибутов конкретных столбцов, составные указываются отдельно и могут применяться к нескольким столбцам. Все ограничения могут быть заданы также визуальными средствами – в конструкторе таблиц или схеме данных.

Общая форма оператора:

CREATE TABLE имя_таблицы (
имя_столбца **тип**[**(размер)**] [атрибуты]

[,имя_столбца **тип**[**(размер)**] [атрибуты] …]
[,**PRIMARY KEY** (имя_столбца [, имя_столбца …])]
[,**FOREIGN KEY** (имя_столбца [, имя_столбца …])
REFERENCES имя_таблицы
[(имя_столбца [, имя_столбца])]
[**ON UPDATE CASCADE** | **SET NULL**]
[**ON DELETE CASCADE** | **SET NULL**]
]
)

Некоторые атрибуты, задающие простые ограничения для столбцов:

- **NULL** или **NOT NULL**
  Соответственно разрешает или запрещает помещать пустые значения в этот столбец.
- **UNIQUE**
  Накладывает требование уникальности значений. Для любых двух строк таблицы значения в этом столбце не могут быть одинаковыми.
- **PRIMARY KEY**
  Объявляет столбец первичным ключом таблицы. Имеет тот же смысл, что и задание ключевого поля в конструкторе таблиц (см. [1]).
- **REFERENCES** имя_таблицы [(имя_столбца)]
  [**ON UPDATE CASCADE** | **SET NULL**]
  [**ON DELETE CASCADE** | **SET NULL**]
  Объявляет данный столбец внешним ключом, который связан с заданным полем заданной таблицы. Поле, с которым устанавливается связь, должно быть ключевым, или, по крайней мере, уникальным.
  Необязательные атрибуты **ON UPDATE** и **ON DELETE** могут принимать значения **CASCADE** или **SET NULL**.
  Задание ограничения **REFERENCES** имеет тот же смысл, что и создание связи между таблицами с использованием **схемы данных** (см. [1]).
  Использование атрибутов **ON UPDATE** и **ON DELETE** равносильно заданию соответствующих свойств этой связи («Обеспечение целостности данных», «Каскадное обновление связанных полей», «Каскадное удаление связанных записей») в схеме данных.

Пример 1.26.

CREATE TABLE МояТаблица (
Поле1 **COUNTER PRIMARY KEY**,
Поле2 **INTEGER UNIQUE NOT NULL**,
Поле3 **CHAR NOT NULL**
);

Рассмотрим теперь составные ограничения:

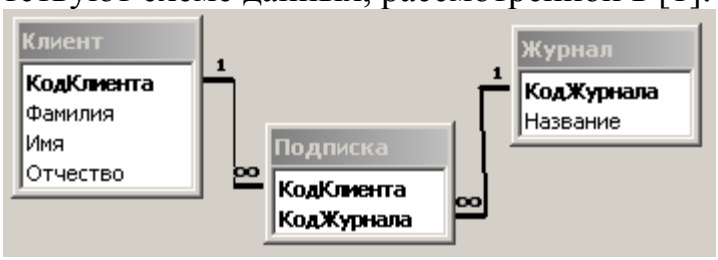
- **PRIMARY KEY** (имя_столбца [, имя_столбца …])
  Объявляет первичный ключ таблицы. В скобках перечисляются столбцы, входящие в первичный ключ.

- **FOREIGN KEY** (имя_столбца [, имя_столбца …]) **REFERENCES** имя_таблицы [(имя_столбца [, имя_столбца …])] [**ON UPDATE CASCADE** | **SET NULL**] [**ON DELETE CASCADE** | **SET NULL**]
  Объявляет столбцы, перечисленные в скобках, внешним ключом. Предложение **REFERENCES** указывает имя таблицы и ее столбцов, с которыми устанавливается связь.

Пример 1.27. Ниже приведены операторы, создающие три связанные таблицы:

```
CREATE TABLE Клиент (
    КодКлиента COUNTER PRIMARY KEY,
    Фамилия CHAR NOT NULL,
    Имя CHAR NOT NULL,
    Отчество CHAR NOT NULL
);
CREATE TABLE Журнал (
    КодЖурнала COUNTER PRIMARY KEY,
    Название CHAR NOT NULL
);
CREATE TABLE Подписка (
    КодКлиента INTEGER REFERENCES Клиент(КодКлиента),
    КодЖурнала INTEGER REFERENCES Клиент(КодЖурнала),
    PRIMARY KEY (КодКлиента, КодЖурнала)
);
```

Таблицы соответствуют схеме данных, рассмотренной в [1]:



Оператор DROP TABLE

DROP TABLE имя_таблицы

Удаляет таблицу с заданным именем.

2. Основы проектирования баз данных

Рассмотренные здесь методы проектирования и анализа моделей баз данных не ориентированы жестко на Access и могут использоваться применительно к любым реляционным СУБД. Вопросы построения таблиц и связей собственно в реляционной модели, реализованной в Access, были рассмотрены в разделе 1.2 и в [1].

Проектирование – важный этап разработки БД. При разработке базы данных необходимо сначала построить и проанализировать модель данных, которые предполагается хранить в БД.

Ошибки, допущенные при проектировании, могут привести в процессе эксплуатации базы данных к различным **аномалиям**, нарушающим ее работоспособность.

Ниже перечислены некоторые возможные последствия ошибок проектирования:

- **Избыточность** (дублирование) данных.
  Например, в информацию о каждом заказе включается имя клиента.
- **Аномалии обновления, противоречивость** (несогласованность) данных, как следствие их избыточности.
  Если в одном из заказов имя клиента введено с ошибкой, его невозможно соотнести с другими заказами этого клиента. С точки зрения СУБД это уже другой клиент.
  Если нужно изменить некоторое дублирующееся значение, понадобится заменить все его вхождения в БД, чтобы избежать противоречий.
- **Аномалии добавления** – невозможность разместить данные о некотором объекте без внесения лишних (или ложных) данных.
  Невозможно добавить в БД клиента, который не связан с каким-либо заказом.
- **Аномалии удаления** – невозможность удалить устаревшую информацию без удаления той, которая может быть еще актуальна.
  При удалении информации обо всех заказах клиента будет утрачена информация и о самом клиенте.

Для разработки и анализа баз данных используются два вида моделей:

- **Инфологическая модель**.
  Она предназначена для описания предметной области.
  В качестве инфологической модели далее будет использоваться модель «**сущность-связь**».
- **Даталогическая модель**.
  Модель описывает структуру данных и их взаимодействия в терминах, пригодных для представления в СУБД. В роли даталогической модели будет выступать **реляционная** модель данных.

Инфологическая модель данных предназначена для описания объектов и выявления их взаимосвязей и может быть построена независимо от даталогической модели, которая будет использована впоследствии.

Задача проектирования состоит в разработке инфологической модели данных и преобразовании ее в адекватную даталогическую.

2.1. Модель «сущность-связь»

Сущности и их атрибуты

Сущность – некоторый объект, явление из рассматриваемой предметной области. Примеры сущностей: человек, автомобиль, сделка, визит к стоматологу.

Атрибуты – данные, описывающие свойства сущности. Примеры атрибутов: фамилия, цвет, стоимость, дата.

В контексте различных предметных областей одно и то же явление может считаться как сущностью, так и атрибутом.

Следует различать **тип** сущности и конкретные ее **экземпляры**.

Тип сущности – признак принадлежности к некоторому классу (множеству) объектов (явлений). Тип сущности характеризуется множеством ее атрибутов.

Экземпляр сущности – конкретный объект, принадлежащий определенному классу объектов. Экземпляр сущности определяется значениями ее атрибутов.

Фактически, в базе данных хранятся только наборы значений атрибутов. Каждый такой набор определяет один экземпляр сущности.

Пример 2.1.

Сущность – «персона» Атрибуты:	Экземпляр сущности «персона» Значения атрибутов:
Фамилия	Иванов
Имя	Иван
Отчество	Иванович
Дата рождения	15.05.1967
Номер паспорта	40 03 012345
Номер телефона	123-45-67, 987-65-43

Обратим внимание, что атрибут «номер телефона» может иметь более одного значения.

Многозначные атрибуты. Может оказаться, что некоторый атрибут сущности способен принимать одновременно несколько значений. Реляционная модель, в которую должна быть впоследствии преобразована данная инфологическая модель, не допускает многозначности атрибутов. Данное противоречие должно быть разрешено. Возможное решение – образование новой сущности.

В приведенном ранее примере атрибут «номер телефона» становится кандидатом на создание сущности «номер телефона».

Другое решение – заменить название атрибута «**номер телефона**» на «**перечень номеров телефонов**» и позволить хранить несколько номеров, перечисленных через запятую, в виде одной текстовой строки. Кроме того, некоторые СУБД позволяют размещать в ячейках таблиц **массивы**, содержащие несколько элементов. Однако при сохранении многозначного атрибута могут возникнуть сложности с извлечением данных. Например, будет затруднительно составить запрос следующего вида: «извлечь из БД сведения о персонах, имеющих заданный номер телефона».

Кроме того, возможное решение зависит от общего контекста разрабатываемой БД. Если речь идет о контактных номерах телефонов сторонних клиентов, номер, скорее всего, является атрибутом и возможные повторы маловероятны и несущественны. Если же ставится задача построения телефонного справочника некоторой организации, в которой, с одной стороны, один сотрудник может быть доступен по нескольким номерам, а с другой – по одному номеру может быть доступно несколько сотрудников, номер телефона, очевидно, становится самостоятельной сущностью.

Домен атрибута – множество значений, которые атрибут может принимать. Доменом может быть, например, множество допустимых значений даты, диапазон целых чисел или множество текстовых строк. Также это может быть множество цветов и оттенков или список компаний - поставщиков.

При реализации модели на языке конкретной СУБД домены приводятся в соответствие с имеющимися типами данных.

Если в процессе разработки и анализа модели выявляются достаточно специфические домены (например - «множество сотовых операторов»), они могут рассматриваться в качестве кандидатов на создание новых сущностей.

Идентификация сущностей. Для того чтобы экземпляры сущности можно было отличить один от другого, следует описать способ их идентификации.

Роль идентификатора выполняет атрибут или группа атрибутов, совокупность значений которых уникальна для каждого экземпляра сущности. В реляционной модели такой уникальный идентификатор называют **первичным ключом**.

Во многих случаях для идентификации может быть использован специальный целочисленный атрибут (номер экземпляра сущности).

Связи между сущностями. После определения сущностей следует описать связи (взаимоотношения) между ними.

Связь устанавливается между **экземплярами** сущностей, а не их типами. Например, для сущностей «клиент» и «заказ» связь устанавливается между конкретным клиентом и его заказами. Для связывания экземпляров сущностей используются их уникальные идентификаторы экземпляров сущности (первичные ключи).

Виды связей:

- **Один-к-одному**
- **Один-ко-многим**
- **Многие-ко-многим**

Вид связи определяется количеством экземпляров сущностей, участвующих в связи с каждой из сторон.

При связи вида «один-к-одному», экземпляр каждой из сущностей может быть связан не более чем с одним экземпляром другой сущности.

Возможное применение связи «один-к-одному» – хранить отдельно некоторый набор секретных атрибутов, для доступа к которым нужны более высокие привилегии. Пример – связь между сущностями «клиент» и «кредитная карта».

При обнаружении связи вида «один-к-одному» следует проанализировать причину ее возникновения. Часто такую связь лучше преобразовать в «один-ко-многим» или уничтожить, объединив две сущности в одну (новая сущность будет содержать атрибуты обеих первоначальных сущностей).

Пример связи «один-ко-многим» – связь между клиентом и его заказами. Клиент может сделать много заказов (или ни одного), но заказ может быть связан только с одним клиентом. Другой пример: в кошельке может одновременно находиться много денежных банкнот, но каждая банкнота в определенный момент времени может находиться не более, чем в одном кошельке.

Между приведенными примерами есть отличие: заказ обязательно должен быть связан с 1 клиентом (без клиента он не существует), а банкнота может быть связана с 0 кошельков или с 1 кошельком (она способна существовать без кошелька).

«Один-ко-многим» - самый распространенный вид связи в реляционных базах данных.

При связи между сущностями вида «многие-ко-многим» каждому экземпляру первой сущности могут быть поставлены в соответствие несколько экземпляров второй сущности (и наоборот).

Пример – связь между журналами и подписчиками. Подписчик может выписать несколько журналов и каждому наименованию журнала могут соответствовать много подписчиков.

Реляционная модель баз данных не позволяет реализовать связь «многие-ко-многим». Эта проблема может быть решена введением дополнительной сущности (см. ниже).

ER-диаграммы. Наименование диаграмм происходит от английского «**E**ntity-**R**elationship» («сущность-связь»).

Классическая форма представления сущностей и их атрибутов использует прямоугольники для обозначения самих сущностей и овалы – для обозначения их атрибутов. Иногда в диаграммах атрибуты отдельных сущностей опускают, чтобы более очевидной была структура в целом. Также зачастую удобным оказывается применение компактной формы описания сущностей.

Пример. 2.2. На рис. 2.1 представлена рассмотренная ранее сущность «Персона» в двух формах – классической и компактной. Символом «*» помечен атрибут, являющийся идентификатором сущности (иногда вместо использования звездочки, имя такого атрибута подчеркивают).

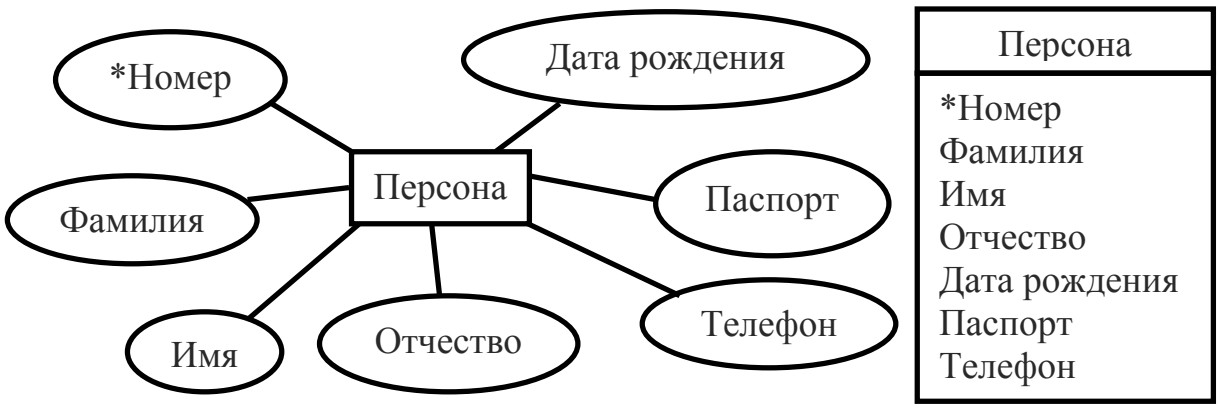


Рис. 2.1. Две формы представления сущности

В классической форме представления ER-диаграмм связи между сущностями принято обозначать ромбом.

Пример 2.3. На рис. 2.2 изображена диаграмма, описывающая две сущности и связь между ними. Пометки «1» и «N» на концах указывают на то, что тип связи - «один ко многим».

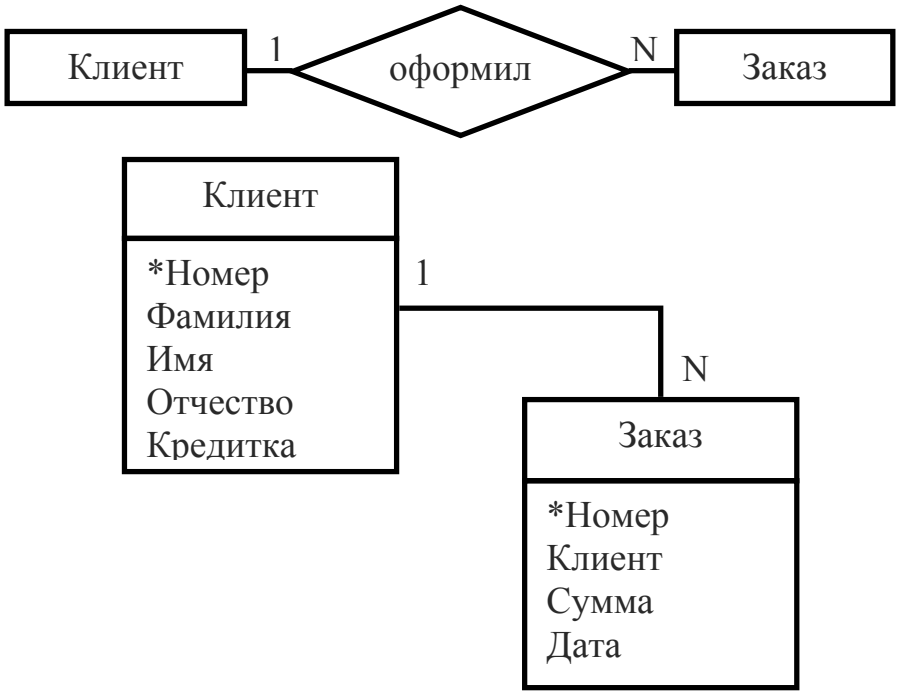


Рис. 2.2. Связи между сущностями

В приведенном примере экземпляр клиента может существовать независимо от наличия заказов. Напротив, заказ обязательно должен быть связан с клиентом.

Сущности, для которых связи являются обязательными, называют **слабыми**. В случае если при проектировании необходимо отметить слабую сущность, ее принято обозначать прямоугольником со сдвоенными границами (рис. 2.3).

Рис. 2.3. Обозначение слабой сущности

Преобразование связи «многие-ко-многим». Связь вида «многие-ко-многим» (рис. 2.4) нереализуема в реляционной модели данных.

Рис. 2.4. Связь «многие-ко многим»

Такую связь требуется преобразовать в две связи вида «один ко многим» введением некоторой вспомогательной сущности.

Рис. 2.5. Преобразованная связь

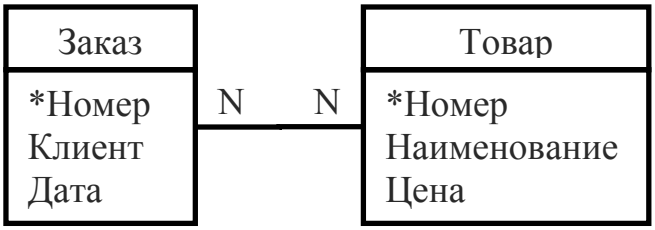
Сущность, предназначенная для образования связи между другими сущностями, называется **составной**.

Составная сущность иногда обозначается ромбом, вписанным в прямоугольник.

Уникальный идентификатор составной сущности образуется сочетанием идентификаторов связанных с ней сущностей.

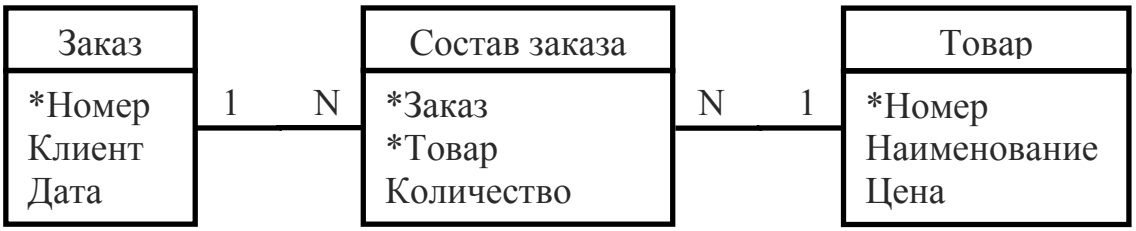
Кроме собственно образования множественной связи, составная сущность может хранить дополнительные данные, относящиеся к этой связи (атрибуты этой связи).

Пример 2.4. Есть сущность «Товар», описывающая имеющиеся товары и сущность «Заказ», регистрирующая заказы, оформленные на эти товары.



В каждый заказ может быть включено много видов товаров. Каждый вид товара может входить во многие заказы. Также необходимо где-то хранить информацию о количестве позиций каждого товара, входящего в заказ. Эта информация не может являться атрибутом заказа. В заказ может входить много товаров и для каждого требуется хранить его количество. Как уже было сказано, многозначность атрибутов не допускается. Количество также не может быть атрибутом товара, поскольку товары существуют независимо от заказов.

Решение проблемы – введение составной сущности «**Состав заказа**».



Каждый экземпляр сущности «Состав заказа» связан только с одним заказом и только с одним видом товара и содержит сведения о количестве единиц данного товара, входящих в данный заказ. Идентификатором экземпляра сущности «Состав заказа» является пара идентификаторов «Товар» и «Заказ».

Анализ предметной области и построение модели. Рассмотрим основные шаги, которые нужно выполнить для построения инфологической модели предметной области. Иногда выявленные на каком-то этапе противоречия могут потребовать повторения уже проделанных действий с учетом новых фактов.

1. **Отбор документов и экспертных знаний, содержащих информацию об объектах, явлениях и процессах, имеющих место в предметной области**.
   Разработчик может не быть экспертом в рассматриваемой предметной области. Эксперт не обязан разбираться в принципах построения баз данных. Анализ документов и общение с экспертом помогают разработчику изучить «инфологический» аспект предметной области.
2. **Выявление сущностей и их атрибутов**.
   Иногда сущности изначально ярко выражены. Остается выявить и документировать все их атрибуты. В некоторых случаях наличие сущности можно выявить, лишь изучив различные объекты и установив, что они являются атрибутами одной и той же сущности (наличие которой изначально не было очевидным).
3. **Выявление и анализ зависимостей (взаимосвязей) между различными сущностями или атрибутами**.

На данном этапе может возникнуть необходимость изменения статуса некоторых объектов – преобразования сущности в атрибут или наоборот.

4. **Определение способа идентификации экземпляров каждой сущности**.
Идентификатор должен гарантированно быть уникальным и по возможности не подвергаться изменениям. В роли идентификатора может выступать уже имеющийся атрибут или комбинация атрибутов, если они отвечают этим условиям. В противном случае должен быть добавлен специальный атрибут – идентификатор.

5. **Определение доменов для выявленных атрибутов**.
Анализ доменов помогает обнаружить новые сущности, не выявленные ранее. Если домен достаточно специфичен, он может образовать новую сущность.

Пример 2.5. Воспользуемся приведенным алгоритмом построения модели.

1. В распоряжении разработчика имеются документы, представленные на рис. 2.6.

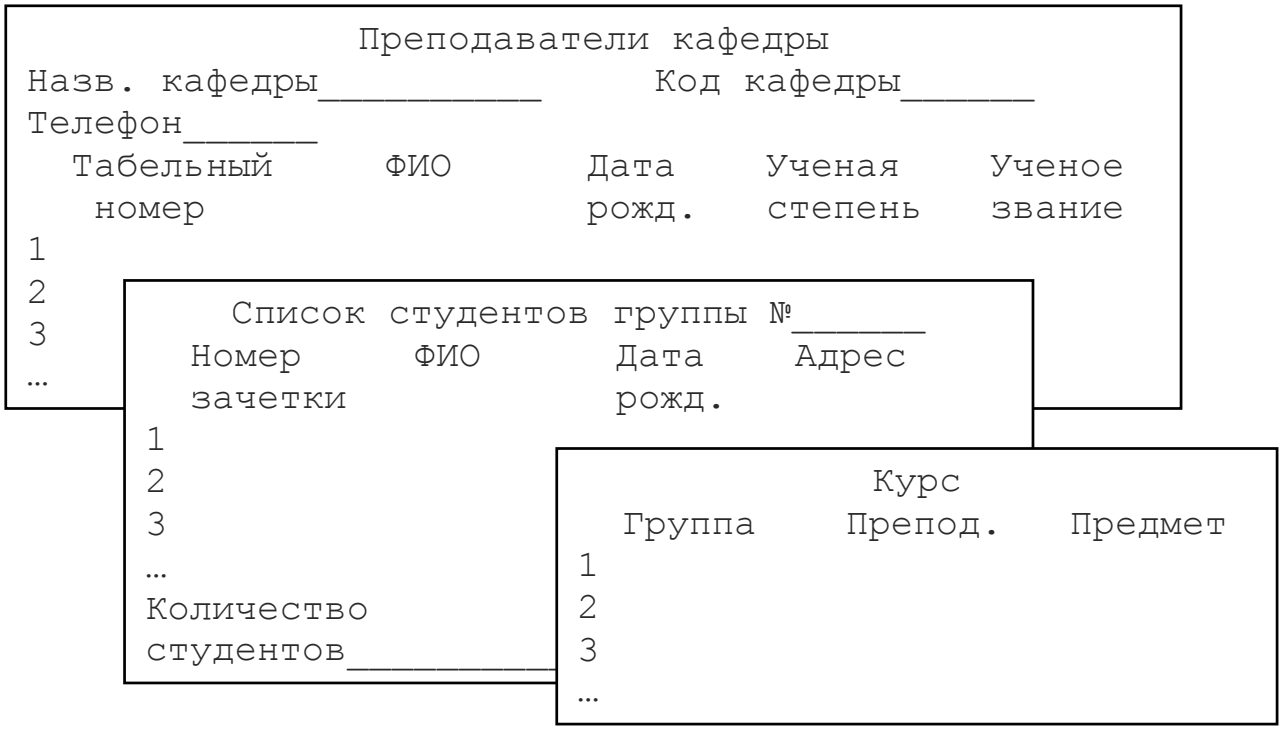


Рис. 2.6. Исходные документы

2. Анализ исходных документов позволяет выявить основные сущности и их атрибуты (рис. 2.7).

Рис. 2.7. Сущности и атрибуты

3. На рис.2.8. изображены выявленные между сущностями связи.

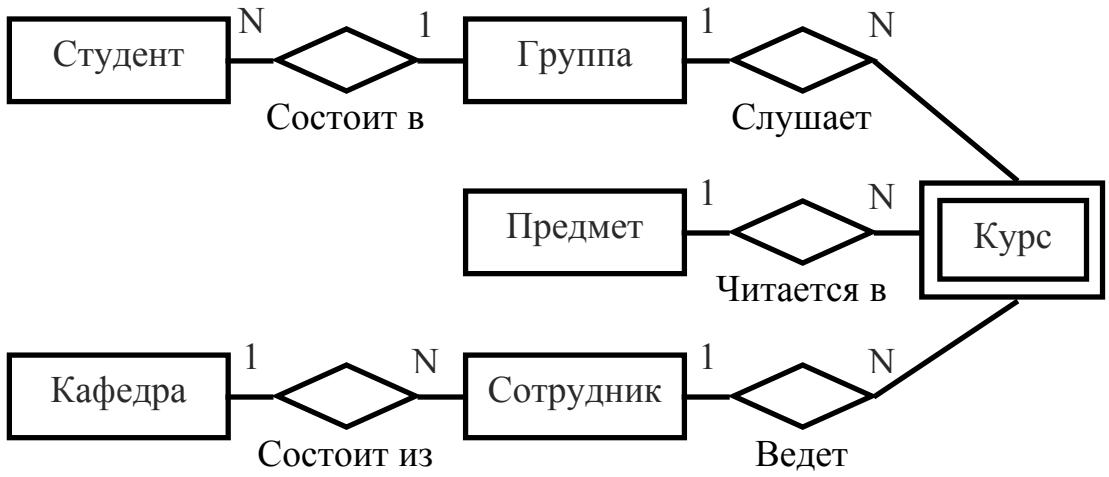


Рис. 2.8. Связи между сущностями

4. На рис. 2.9 показаны сущности и их идентификаторы.

Рис. 2.9. Сущности и их идентификаторы

5. Анализ доменов позволил выявить новые потенциальные сущности – «УченаяСтепень» и «УченоеЗвание» (рис. 2.10).

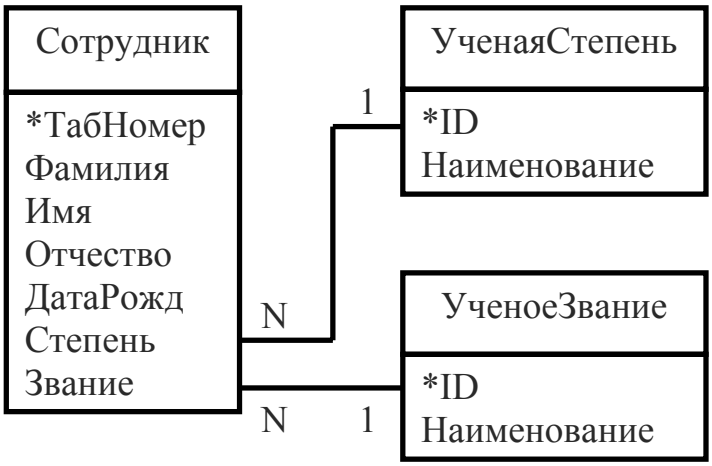


Рис. 2.10. Новые сущности – значения доменов

Модель, полученная в рассмотренном примере, разработана в учебных целях и не охватывает все возможные сущности и виды взаимодействия между кафедрами, преподавателями, студентами и курсами (аудитории, расписание, оборудование и т.п.).

2.2. Реляционная модель данных

Начальные сведения о реляционной модели данных были даны в [1]. Здесь мы рассмотрим некоторые аспекты этой модели более детально.

Основной элемент реляционной модели – **таблица** (в реляционной терминологии синоним термина «таблица» - отношение). Таблица образуется **столбцами** (атрибутами) и **строками** (кортежами).

Таблицы могут быть **базовыми** или **виртуальными**. Базовые – таблицы, хранящиеся в БД постоянно. Множество базовых таблиц составляет **схему БД**. Виртуальные таблицы возникают как результат действий по манипулированию данными. Виртуальные таблицы существуют непродолжительное время.

Свойства столбцов:

- Имя столбца уникально в пределах одной таблицы.
  Если при манипулировании с данными имена столбцов различных таблиц совпадают, к имени столбца добавляется префикс из имени таблицы и точки, например: *Студент.Фамилия*
- Значения столбца принадлежат только одному домену.
  При этом говорят, что таблицы **однородны по столбцам**.
  СУБД должна обеспечивать соблюдение этого свойства, позволяя размещение в столбцах только значений из разрешенного домена.

Свойства строк:

- На пересечении столбца и строки находится только одно значение.
  Как уже было сказано, не допускается наличие многозначных атрибутов.

- Строки в таблице должны быть уникальными.
  Если некоторые строки полностью совпадают, невозможно идентифицировать одну из них.
- Наличие первичного ключа.
  Первичный ключ – столбец или комбинация столбцов, однозначно определяющих каждую строку таблицы. Наличие первичного ключа гарантирует уникальность строк.

Для виртуальных таблиц требование уникальности строк и наличия первичных ключей могут не соблюдаться

Первичные и внешние ключи. Единственный способ описания связей между данными в реляционной модели – использование первичных и внешних ключей. Использование значения первичного ключа некоторой таблицы в качестве внешнего ключа в другой таблице образует связь между соответствующими строками этих таблиц

Первичный ключ – наименьший (несократимый) набор столбцов таблицы, способных обеспечить уникальность каждой строки. Первичный ключ не может содержать пустое значение (Null). Чаще первичный ключ состоит из одного столбца, но не всегда.

Иногда, особенно для составных сущностей, бывает нецелесообразно вводить специальный столбец для образования первичного ключа.

Составной первичный ключ – ключ, состоящий из нескольких столбцов. Уникальной является **комбинация значений** всех столбцов ключа. Каждый столбец ключа сам по себе может не быть уникальным.

Внешний (вторичный) ключ – столбец, значения которого должны совпадать с первичным ключом некоторой таблицы. Иными словами, для внешнего ключа доменом является множество значений первичного ключа. Внешний ключ не может принимать значения, не содержащиеся в первичном ключе. Единственно возможное исключение – значение Null (пусто), при котором внешний ключ ни на что не ссылается. Поддержание целостности ключей является задачей СУБД.

Допустимой является ситуация, когда содержащийся в таблице внешний ключ ссылается на первичный ключ той же самой таблицы. Пример: запись с информацией о сотруднике, размещенная в таблице «Сотрудник», имеет атрибут «Начальник», ссылающийся на другую запись в той же самой таблице.

Внешний ключ может быть (или не быть) частью составного первичного ключа в своей таблице.

Документирование таблиц. Основные сведения о таблице – название, набор полей и первичный ключ, могут быть задокументированы кратко в текстовой форме:

ИмяТаблицы(Поле1, Поле2, Поле3, Поле4, …).

Имена полей, входящих в первичный ключ таблицы, подчеркивают. Такая форма описания удобны при анализе структуры таблицы, без учета связей – например, при проведении нормализации.

Переход от инфологической модели к реляционной. Описанные в инфологической модели сущности преобразуются в таблицы, атрибуты сущностей становятся столбцами этих таблиц.

Домены атрибутов должны быть приведены в соответствие с имеющимися в СУБД типами данных. Некоторые СУБД позволяют разработчику формировать собственные домены (типы данных).

Определенные для сущностей уникальные идентификаторы становятся первичными ключами таблиц.

Связи между таблицами вида «один-ко-многим» (одной записи главной таблицы может соответствовать несколько записей в связанной) строятся добавлением в связанные таблицы внешних ключей, ссылающихся на первичные ключи главных таблиц.

Для создания связи «один-к-одному», поле внешнего ключа должно быть первичным ключом связанной таблицы, либо на него должно быть наложено условие уникальности значений.

Каждая связь вида «многие-ко-многим» должна быть преобразованы в две связи «один-ко-многим».

2.3. Нормализация

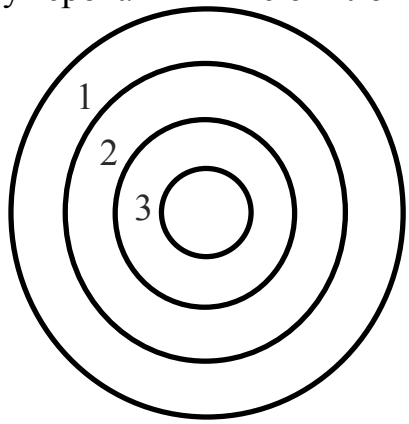
Процесс нормализации – приведение структуры базы данных в соответствие с некоторым набором формальных правил.

Сущности и атрибуты, выявленные в процессе моделирования предметной области, могут быть преобразованы в таблицы (отношения) и связи различными способами. Требуется выявить способы, наиболее адекватные реляционной модели. Нужно убедиться, что созданная реляционная модель не порождает противоречий в данных.

Нормализация позволяет устранить недостатки в реляционной структуре данных и устранить **аномалии** проектирования. Нормализация особенно актуальна для больших проектов, содержащих множество сущностей с большим числом взаимосвязей.

Для проверки правильности разработанной структуры применяют так называемые **нормальные формы** – правила, которым должны отвечать отношения (таблицы).

Нормальные формы пронумерованы и имеют вложенную взаимосвязь.



Это означает, что, если отношение соответствует некоторой нормальной форме, оно также соответствует нормальным формам с меньшими номерами.

Первая нормальная форма (1НФ). Таблица соответствует **1НФ**, если в ней отсутствуют повторяющиеся группы.

Повторяющаяся группа – столбец, имеющий несколько значений в каждой строке. Повторяющиеся группы в реляционной модели соответствуют **многозначным атрибутам** в модели «**сущность-связь**». Пример многозначного атрибута рассматривался выше. Решение проблемы – создание новой таблицы (сущности) для хранения экземпляров из повторяющейся группы.

Вторая нормальная форма (2НФ). Таблица соответствует **2НФ**, если она соответствует **1НФ** и все атрибуты, не входящие в первичный ключ, связаны с ним **полной функциональной зависимостью**.

Атрибут **B** функционально зависит от атрибута **A** той же таблицы, если в любой заданный момент времени для каждого из различных значений поля **A** обязательно существует только одно из различных значений поля **B** (иначе говоря, если известно **A**, можно однозначно установить **B**).

Пример 2.6. Наличие функциональной зависимости. Пусть имеется следующая таблица:

Персона(Номер, Фамилия, Имя, Отчество, ДатаРождения)

Очевидно, что существует функциональная зависимость атрибутов:

Номер → Фамилия, Имя, Отчество, ДатаРождения

Полная функциональная зависимость: все атрибуты зависят от составного ключа и не зависят ни от какой его части.

Пример 2.7. Нарушение условия **2НФ**. Дана таблица:

Кафедра(КодИнститута, КодКафедры, Название, Телефон, Адрес)

Здесь составной ключ - «КодИнститута, КодКафедры» (поле «КодКафедры» идентифицирует кафедру внутри института).

Имеет место функциональная зависимость:

КодИнститута, КодКафедры → Название, Телефон, Адрес

Для каждой кафедры указывается адрес. Однако, скорее всего, адрес на самом деле зависит только от кода института (т.е. от части ключа).

Для исправления ситуации адрес должен стать атрибутом института. Если же институт имеет несколько площадок (корпусов), у каждой из которых собственный адрес, можно ввести новую сущность (таблицу) «Корпус» и для каждой кафедры указывать корпус, в котором она размещается.

Третья нормальная форма (3НФ). Таблица соответствует **3НФ**, если она соответствует **2НФ** и не существует **транзитивных зависимостей**.

Если A → B и B → C, то A → C (C зависит от A транзитивно).

Пример 2.8. Нарушение условий **3НФ**.

Кафедра(КодКафедры, Название, Телефон, Корпус, Адрес)

В отличие от примера 2.7, первичный ключ состоит только из одного атрибута. Следовательно, все неключевые атрибуты связаны с ним полной функциональной зависимостью (условие **2NF** соблюдено). Однако здесь адрес также не на своем месте, как и в предыдущем случае. Можно выделить две зависимости:

КодКафедры → Название, Телефон, Корпус

Корпус → Адрес

Адрес определяется корпусом, в котором размещается кафедра. Следовательно, имеет место транзитивная зависимость:

КодКафедры → Корпус → Адрес

Другие нормальные формы.

- Нормальная форма **Бойса-Кодда (НФБК)** [8 – 13]. Устраняет некоторые недостатки третьей нормальной формы, принимая к рассмотрению различные потенциальные ключи (столбцы или группы столбцов, которые могли бы выступить в роли первичного ключа).
- **Четвертая** нормальная форма **(4НФ)** [8, 9, 11 – 13]. Выявляет многозначные зависимости. Многозначной зависимостью называется ситуация, когда одному и тому же значению из некоторого столбца **A** соответствует несколько значений в столбце **B** и несколько значений в столбце **C**, при этом значения в столбцах **B** и **C** не связаны между собой.
- **Пятая** нормальная форма **(5НФ)** [8, 11, 13]. Рассматривает более сложные случаи многозначных зависимостей.
- **Доменно-ключевая** нормальная форма **(ДКНФ)** [12]. Определяет необходимое и достаточное условие отсутствия аномалий. Поскольку в настоящее время не известен алгоритм преобразования таблиц к этой форме, она не представляет большого практического интереса для разработчиков.

Приведенные сведения о процессе нормализации неполны, за подробностями можно обратиться к литературе ([8 – 13]). Однако для разработки большинства реляционных проектов обычно бывает достаточно первых трех нормальных форм.

Приложение 1. Справочные сведения

Ниже приведены некоторые сведения, которые могут быть полезны при составлении запросов в Access. За более подробной информацией следует обращаться к источникам [2 – 4].

Таблица П.1.1. Типы данных Access

Название	Описание
Char	Текстовое поле. Может иметь размер не более 255 символов.
Text, Memo	Текст большого размера (может вместить более 1 млрд. символов).
Logical	Логический тип. Может принимать одно из двух значений: True или False.
Byte	Целое в диапазоне от 0 до 255.
Short	Целое в диапазоне от -32768 до +32767.
Integer, Int, Long	Длинное целое (в диапазоне от -2147483648 до 2147483647).
Single	Число с плавающей точкой одинарной точности. Может принимать значения в диапазоне от -3.4×10^{38} до 3.4×10^{38}.
Double, Number	Число с плавающей точкой двойной точности. Может принимать значения в диапазоне от -1.8×10^{308} до 1.8×10^{308}.
Date, Time, DateTime	Дата и время.
Currency	Используется для обозначения денежных сумм. Запоминаются 11 знаков слева от десятичной точки и 4 знака справа от десятичной точки.
Counter	Длинные целые с автоматическим приращением.
OLEObject	OLE-объекты, созданные в других программах с использованием протокола OLE. Размер – до 2 Гбайт.
Binary	Любой двоичный объект размером до 2 Гбайт. Этот тип может быть использован, например, для хранения двоичных файлов.

Таблица П.1.2. Некоторые групповые операции

Имя операции	Описание
Sum	Вычисляет сумму полей в группе.
Avg	Вычисляет среднее значение для полей группы.
Min	Находит наименьшее значение в группе.
Max	Находит наибольшее значение в группе.
Count	Подсчитывает количество элементов в группе. В качестве аргумента можно использовать «*».
First	Возвращает первое значение из группы
Last	Возвращает последнее значение из группы

Таблица П.1.3. Функции обработки текста

Функция	Описание
Left(строка, n)	Возвращает n левых символов строки.
Right(строка, n)	Возвращает n правых символов строки.
Mid(строка, n1, n2)	Возвращает n2 символов строки, начиная с позиции n1.
InStr(строка1, строка2)	Номер позиции, с которой строка2 входит в строка1.
Ltrim(строка)	Удаляет пробелы из начала строки.
Rtrim(строка)	Удаляет пробелы из конца строки.
Trim(строка)	Удаляет пробелы из начала и конца строки.

Таблица П.1.4. Функции обработки даты и времени

Функция	Описание
Date()	Возвращает текущую дату.
Now()	Возвращает текущую дату и время.
DateDiff(интервал, дата1, дата2)	Определяет разницу между датами. Аргумент «интервал» определяет способ представления разницы:

	“yyyy” – год, “q” – квартал, “m” – месяц, “y” – день года, “d” – день, “w” – неделя, “h” – час, “n” – минута, “s” – секунда.
DateAdd(интервал, число, дата)	Будущая дата, отстоящая от указанной на заданное число интервалов.
Year(дата)	Возвращает число - значение года для указанной даты.
Month(дата)	Возвращает число - значение месяца для указанной даты.
Day(дата)	Возвращает число - значение дня для указанной даты.

Таблица П.1.5. Функции преобразования

Функция	Описание
Str(аргумент)	Преобразует значение аргумента в текстовую строку
Val(строка)	Преобразует строку в число
Int(число)	Возвращает целую часть числа

Таблица П.1.6. Операции

Операция	Описание
+	Сложение, конкатенация строк
-	Вычитание
*	Умножение
/	Деление
=	Равно
<>	Не равно
>	Больше
<	Меньше
>=	Больше или равно

<=	Меньше или равно
AND	Логическое «И»
OR	Логическое «ИЛИ»
NOT	Логическое отрицание
операция ANY подзапрос	Проверка на соответствие условию любого элемента из подзапроса
операция ALL подзапрос	Проверка на соответствие условию всех элементов из подзапроса
EXISTS подзапрос	Проверка на существование в подзапросе хотя бы одного элемента
аргумент IS NULL	Является ли аргумент пустым значением
аргумент IS NOT NULL	Является ли аргумент непустым значением
аргумент1 BETWEEN аргумент2 AND аргумент3	Находится ли значение аргумента «аргумент1» между значениями «аргумент2» и «аргумент3»
аргумент LIKE образец	Проверка совпадения аргумента с образцом. В образце может присутствовать символ «%», обозначающий любое количество любых символов. Например, выражение ’Access’ LIKE ’A%s’ должно возвращать значение «True» (истина)

Задание 1. Создание запросов на языке SQL (часть 1, знакомство)

Цель задания: Освоить составление запросов на языке SQL при создании объектов базы данных. Используется разработанная ранее база данных «Журналы».

Последовательность действий:

1. Запустите Microsoft Access и откройте базу данных «Журналы».
2. Создайте запрос с именем «Самый простой запрос», который вычисляет значение выражения «2+2» и выдает текущее время, не используя при этом таблицы или другие запросы базы данных.

- Щелкните по ярлыку «**Создание запроса в режиме конструктора**».
- Закройте диалоговое окно «**Добавление таблиц**».
- Перейдите в режим SQL ().
- В окне редактирования введите текст запроса «**SELECT 2+2, NOW();**».
- Перейдите в режим просмотра результатов работы запроса  . Обратите внимание, что названия столбцов сформированы автоматически.
- Перейдите в режим конструктора (). Обратите внимание, какие имена присвоил конструктор столбцам запроса.
- Перейдите в режим SQL и назначьте столбцам новые имена: «**ДваПлюсДва**» и «**ДатаИВремя**» соответственно.
- Просмотрите результат работы запроса.
- Закройте окно запроса, сохранив его под именем «**Самый простой запрос**».

3. Создайте в режиме конструктора и модифицируйте в режиме SQL запрос «Журналы с кодами клиентов», выводящий список журналов и коды клиентов, которые на них подписаны.

- Запустите конструктор для создания нового запроса.
- Добавьте в запрос таблицы «**Журнал**» и «**Подписка**».
- Перенесите в бланк запроса имена полей «**Название**» и «**Клиент**».
- Просмотрите результат работы запроса.
  Запрос выдает названия только тех журналов, на которые подписан хотя бы один клиент. Следует изменить способ объединения таблиц с внутреннего на левое внешнее.
  *Обратите внимание, из таблицы «**Подписка**» извлекаются именно коды клиентов, а их фамилии отображаются в результирующей таблице благодаря тому, что для таблицы «**Подписка**» были настроены параметры подстановки.*
- Перейдите в режим SQL.

- Замените в предложении «**FROM**» ключевые слова «**INNER JOIN**» на «**LEFT JOIN**».
- Просмотрите результат работы запроса.
  *Теперь в столбце «**Название**» присутствуют названия и тех журналов, у которых нет ни одного подписчика. Для таких журналов в столбец «**Клиент**» помещено пустое значение.*
- Перейдите в режим конструктора. Обратите внимание на то, что в конструкторе способ объединения также изменился.
  Линия связи, обозначающая объединение таблиц, теперь снабжена стрелкой на одном конце.
- Для того, чтобы сохранить запрос, не закрывая окна конструктора, нажмите кнопку «**Сохранить**»  на панели инструментов Access. В диалоговом окне введите имя запроса «**Журналы с кодами клиентов**» и нажмите кнопку «**OK**».
- Выясните, как изменится SQL – представление запроса, если связь между таблицами будет отсутствовать. Нажмите правой кнопкой мыши на линии связи и в контекстном меню выберите пункт «**Удалить**».
  В случае, когда связь между полями таблиц не определена, запрос формирует декартово произведение таблиц.
- Перейдите в режим SQL.
  *Теперь в предложении «**FROM**» вместо оператора объединения «**LEFT JOIN**» с условием равенства значений столбцов двух таблиц, используется оператор «**запятая**», объединяющий записи таблиц безусловно, в виде декартова произведения.*
- Закройте окно запроса, при этом **откажитесь от сохранения последних изменений**.

4. Создайте в режиме SQL запрос «Журналы с количеством клиентов», выводящий список журналов и количество клиентов, которые подписаны на каждый из них.

- Запустите конструктор для создания нового запроса.
- Закройте окно добавления таблиц. Перейдите в режим SQL.
- Введите в окне редактирования запрос следующего вида:

```
SELECT Журнал.Название,
       Count(Подписка.Журнал) AS Количество
FROM Журнал LEFT JOIN
       Подписка ON Журнал.КодЖурнала = Подписка.Журнал
GROUP BY Журнал.Название
ORDER BY Журнал.Название;
```

- Просмотрите результат работы запроса.
- Просмотрите запрос в режиме конструктора. Найдите соответствие между всеми элементами, размещенными в бланке и области таблиц конструктора запроса, и элементами запроса на языке SQL.
- Закройте окно запроса, сохранив запрос под именем «**Журналы с количеством клиентов**».

5. В форме «Подписчики - подчиненная» в качестве источника записей используется специальный запрос «Клиенты с кодами журналов». Измените свойства формы таким образом, чтобы вместо внешнего запроса использовался запрос на языке SQL, заданный непосредственно внутри формы.

- Откройте форму «**Подписчики - подчиненная**» в режиме конструктора.
- Просмотрите свойство «**Источник записей**» данной формы.
  *В качестве источника указан запрос «**Клиенты с кодами журналов**».*
- Откройте запрос «**Клиенты с кодами журналов**» в режиме SQL.
- Выделите и скопируйте в буфер обмена текст SQL – запроса.
- Вставьте скопированный текст запроса из буфера обмена в строку свойства «**Источник записей**».
- Просмотрите результат работы формы. Убедитесь, что ее поведение осталось прежним.
- Закройте форму «**Подписчики - подчиненная**», сохранив внесенные изменения. Закройте окно запроса.
- Откройте форму «**Список журналов**», которая использует измененную выше форму в качестве подчиненной. Убедитесь, что форма работает верно.
  *Теперь запрос «**Клиенты с кодами журналов**» больше не нужен. Источник записей формы образуется встроенным запросом на языке SQL.*

Самостоятельная работа:

1. Просмотрите все созданные ранее запросы в режиме SQL. Найдите соответствие между предложениями на языке SQL и элементами запроса, отображаемыми в режиме конструктора.

2. Просмотрите в режиме SQL все запросы базы данных «Борей». Сравните их представление в режиме SQL и режиме конструктора. *Некоторые запросы на языке SQL могут не иметь соответствующего представления в режиме конструктора (**выявите такие запросы**). Средства языка SQL предоставляют более широкие возможности по сравнению с возможностями конструктора.*

3. Просмотрите свойства форм базы данных «Борей». Выявите и изучите все случаи использования встроенных запросов SQL.

Задание 2. Создание запросов на языке SQL (часть 2, углубленное изучение)

Цель задания: Освоить приемы составления запросов с использованием различных операторов языка SQL.

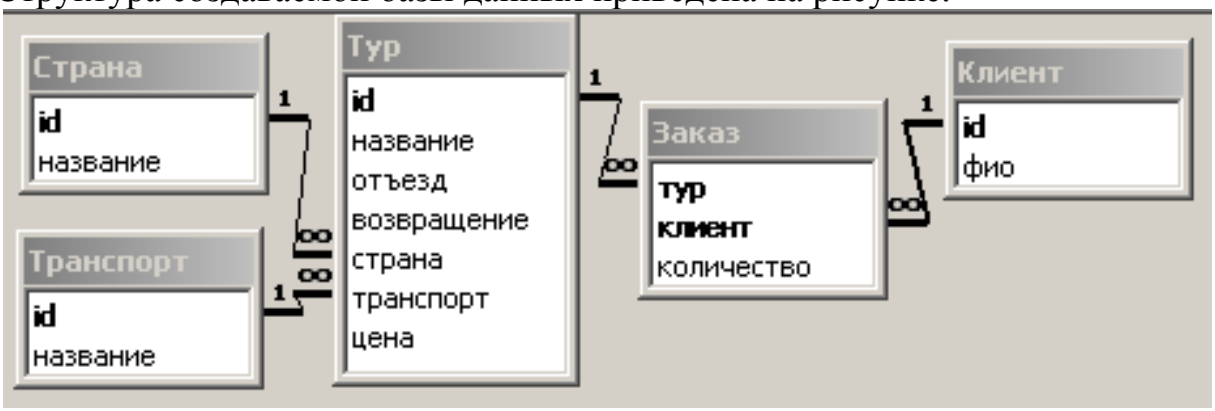
Последовательность действий:

1. Запустите Microsoft Access и создайте новую базу данных с именем «**ТурАгентство**».

База данных должна содержать данные о туристических поездках, клиентах и заказах. Таблицы базы данных следует создавать средствами языка SQL, без использования визуальных инструментов. Для того, чтобы создать таблицу, не-

обходимо сначала создать запрос SQL на создание таблицы. Обычно при создании базы данных нет необходимости создавать все таблицы посредством SQL, но иногда операторы SQL могут быть использованы в приложениях для создания или изменения структуры таблиц автоматически.

Структура создаваемой базы данных приведена на рисунке.



2. Создайте запрос с именем «**СоздатьТблТранспорт**». В режиме SQL введите текст запроса на создание таблицы:

```
CREATE TABLE Транспорт (
  id COUNTER PRIMARY KEY,
  название TEXT(30)
);
```

Этот запрос создает таблицу «**Транспорт**», содержащую два поля:

id – уникальный числовой идентификатор (первичный ключ) записи, значения которого назначаются автоматически (в режиме конструктора таблиц этот тип данных называется «**Счетчик**»);

название – текстовое поле, которое может содержать не более 30 символов. Таблица будет содержать названия видов транспорта (самолет, автобус и т.п.), применяемых при организации туров.

3. Создайте запрос с именем «**СоздатьТблСтрана**», содержащий текст запроса на языке SQL:

```
CREATE TABLE Страна (
  id COUNTER PRIMARY KEY,
  название TEXT(30)
);
```

Этот запрос создает таблицу «**Страна**». Таблица будет содержать список стран, в которые могут организовываться туры.

4. Создайте запрос с именем «**СоздатьТблТур**». В режиме SQL введите текст запроса:

```
CREATE TABLE Тур (
  id COUNTER PRIMARY KEY,
  название TEXT(30),
  отъезд DATE,
  возвращение DATE,
  страна INTEGER REFERENCES Страна (id),
  транспорт INTEGER REFERENCES Транспорт (id),
```

```
  цена CURRENCY NOT NULL
);
```

Запрос создает таблицу «**Тур**». Таблица будет содержать сведения о конкретных туристических поездках.

Поля «**отъезд**» и «**возвращение**» будут содержать даты начала и окончания тура соответственно. Для поля «**страна**» с помощью ключевого слова «**REFERENCES**» («ссылается») определяется связь с полем «**id**» таблицы «**Страна**». Аналогично, поле «**транспорт**» связывается с полем «**id**» таблицы «**Транспорт**». Поле цена (имеющее денежный тип данных) предназначена для хранения информации о стоимости тура и не может быть пустым.

5. Создайте запрос с именем «**СоздатьТблКлиент**», содержащий текст запроса:

```
CREATE TABLE Клиент (
  id COUNTER PRIMARY KEY,
  фио TEXT(50)
);
```

Таблица «**Клиент**» будет содержать имена клиентов, которые обращались в данное туристическое агентство. Текстовое поле «**фио**» может содержать до 50 символов.

6. Создайте запрос с именем «**СоздатьТблЗаказ**»:

```
CREATE TABLE Заказ (
  тур INTEGER REFERENCES Тур (id),
  клиент INTEGER REFERENCES Клиент (id),
  количество INTEGER NOT NULL,
  PRIMARY KEY (тур, клиент)
);
```

Таблица «**Заказ**» будет содержать сведения о заказах, оформленных клиентами данного турагентства на организуемые им туристические поездки. Поле «**тур**» связывается с полем «**id**» таблицы «**Тур**», поле «**клиент**» - с полем «**id**» таблицы «**Клиент**». Поле «**количество**» показывает, сколько билетов на выбранный тур заказал данный клиент. Пара полей «**тур**» и «**клиент**» образует первичный ключ таблицы. Это означает, что один клиент может заказать несколько билетов на один и тот же тур, но все они должны входить в один заказ (невозможно создать другой заказ на имя того же клиента на тот же самый тур).

7. Выполните созданные запросы в следующей последовательности: «**СоздатьТблТранспорт**», «**СоздатьТблСтрана**», «**СоздатьТблТур**», «**СоздатьТблКлиент**», «**СоздатьТблЗаказ**». Последовательность создания таблиц не может быть произвольной, поскольку поля таблицы «**Тур**» ссылаются на поля таблиц «**Транспорт**» и «**Страна**», а поля таблицы «**Заказ**» - на поля таблиц «**Тур**» и «**Клиент**».

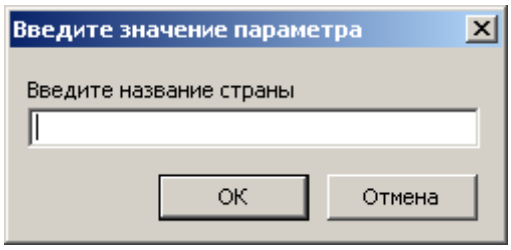
8. Убедитесь, что все необходимые таблицы были созданы. Исследуйте каждую из таблиц с помощью конструктора и выясните, как применение ключевых слов языка SQL (например, «**NOT NULL**» или «**TEXT(30)**») повлияло на свойства полей таблиц.

9. Откройте окно «**Схема данных**» и убедитесь, что между таблицами были установлены все связи, заданные в запросах SQL ключевым словом «**REFERENCES**».

10. Создайте запрос с именем «**ДобавитьСтрану**». В режиме SQL введите текст запроса на добавление данных в таблицу:

```
INSERT INTO Страна
   (название) VALUES ([Введите название страны])
;
```

Этот запрос добавляет в таблицу «**Страна**» новую запись. Запрос содержит параметр, поэтому при его запуске будет выдано окно для ввода дополнительной информации.



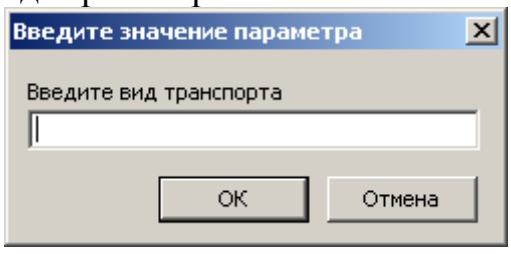
11. Используя созданный запрос, добавьте названия следующих стран:

- Россия
- Турция
- Индия
- Египет
- Околоземная орбита

12. Создайте запрос с именем «**ДобавитьТранспорт**». В режиме SQL введите текст запроса:

```
INSERT INTO Транспорт
   (название) VALUES ([Введите вид транспорта])
;
```

Запрос будет добавлять новые записи в таблицу «**Транспорт**», запрашивая у пользователя название вида транспорта.



13. Используя созданный запрос, добавьте следующие названия видов транспорта:

- Самолет
- Поезд
- Автобус
- Теплоход
- Собачья упряжка
- Слон

- Велорикша
- Верблюд
- Космический корабль

14. Откройте таблицу «**Тур**» в режиме конструктора и определите свойства подстановки столбцов для полей «**страна**» и «**транспорт**», используя данные таблиц «**Страна**» и «**Транспорт**» соответственно.

15. Откройте таблицу «**Тур**» в режиме таблицы и добавьте не менее десяти записей о турах в различные страны и на различных видах транспорта, задавая различные значения для времени отъезда и возвращения, а также цены тура. Каждому туру должно быть присвоено название (по возможности романтическое).

16. Введите в таблицу «**Клиент**» информацию не менее чем о десяти клиентах.

17. Откройте таблицу «**Заказ**» в режиме конструктора и определите свойства подстановки столбцов для полей «**тур**» и «**клиент**», используя название тура из таблицы «**Тур**» и поле «**фио**» из таблицы «**Клиент**» соответственно.

18. Откройте таблицу «**Заказ**» в режиме таблицы и добавьте не менее пятнадцати записей о заказах. Для каждого заказа должно быть указано количество билетов.

19. Самостоятельно создайте запрос «**Суммы по заказам**». Запрос должен выводить список всех заказов (название тура, фио клиента) и для каждого из заказов его стоимость (произведение цены тура на количество заказанных билетов).

20. Самостоятельно создайте запрос «**Суммы по турам**». Запрос должен выводить список всех туров и для каждого тура – стоимость всех проданных на него билетов (вычисляется как произведение цены тура на сумму поля «количество» всех заказов, связанных с данным туром).

21. Создайте запрос «**Суммы заказов по клиентам**». Введите в режиме SQL следующий код:

```
SELECT Клиент.id, Клиент.фио, SUM(Стоимость) AS Сумма
FROM Клиент LEFT JOIN (
  SELECT Тур.цена*Заказ.количество AS Стоимость,
         Заказ.клиент
  FROM Тур INNER JOIN Заказ ON Тур.id = Заказ.тур
) AS ТурЗаказ ON Клиент.id=ТурЗаказ.клиент
GROUP BY Клиент.id, Клиент.фио;
```

Запрос формирует список клиентов и для каждого клиента указывает суммарную стоимость всех его заказов

22. Самостоятельно создайте запрос «**Самый уважаемый клиент**», выводящий имя клиента, который оформил заказов на самую большую сумму.

23. Самостоятельно создайте запрос «**Элитные туры**», выводящий список туров, цена которых выше средней цены по всем имеющимся турам.

Задание 3. Проектирование базы данных

Цель задания: Освоить методы проектирования баз данных. Пройти все этапы проектирования – от анализа документов из предметной области до реализации базы данных в MS Access 2000.

Легенда: Комитет по образованию Вашего города проводит эксперимент, в котором принимают участие несколько школ из различных районов города. Целью эксперимента является изучение возможностей современных информационных технологий применительно к задачам мониторинга и централизованного управления учебным процессом.

Все участвующие в эксперименте школы связаны между собой быстродействующей компьютерной сетью. Требуется создать базу данных, которая будет обеспечивать хранение и обработку данных, относящихся к учебному процессу.

Вам поручено выполнить проектирование структуры базы данных и реализацию пилотной версии с использованием СУБД MS Access.

Исходные документы:

Документ 1. Школы, принимающие участие в эксперименте:

Школа	Район	Адрес	Директор
519	Московский	пр. Юрия Гагарина, 14	Ипатова А.И.
570	Невский	ул. Дыбенко, 23	Круглов В.Н.
335	Невский	ул. Подвойского, 5	Сергеева А.И.
578	Приморский	пр. Художников, 12	Петрова М.Ю.
Гимназия № 303	Красносельский	пр. Ветеранов, 123	Зайцев Ю.Е.

Документ 2. Список классов школы № 519

Класс	Количество учеников	Классный руководитель
10-а	23	Дуров Д.М.
10-б	25	Ипатова А.И.
11-а	22	Малинина К.В.

Документ 3. Список учеников 10-а класса 519 школы (классный руководитель Дуров Д.М.)

Фамилия	Имя	Отчество	Пол
Тарасов	Петр	Юрьевич	М
Питько	Анастасия	Михайловна	Ж
Иваненко	Светлана	Николаевна	Ж
Милюшин	Александр	Сергеевич	М
Шарапов	Артем	Георгиевич	М
Рубаников	Михаил	Владимирович	М

Документ 4. Предметы, преподаваемые в школах

Предмет	Выпускной экзамен
Математика	Обязательный
Физика	Обязательный
Химия	По желанию
Биология	По желанию
Литература	Обязательный
Физкультура	Нет
Иностранный язык*	Обязательный
ОБЖ	Нет

* В некоторых школах может преподаваться два иностранных языка

Документ 5. Преподавание предметов в 10-а классе 519 школы

Предмет	Уроков в неделю	Преподаватель	Годовой экзамен
Математика	3	Ипатова А.И.	Да
Физика	2	Дуров Д.М.	Да
Химия	1	Денисова Т.П.	Да
Биология	1	Фролова М.А.	Да
Литература	2	Малинина К.В.	Да
Физкультура	1	Фролова М.А.	Нет
Французский язык	2	Кострова С.П.	Да
ОБЖ	1	Артамонов В.С.	Нет

Документ 6. Ведомость. Годовые оценки по физике учеников 10-а класса 519 школы. Преподаватель Дуров Д.М.

Ученик	Оценка	Примечание
Тарасов П.Ю.	5	
Питько А.М.	4	
Иваненко С.Н.	5	
Милюшин А.С.	4	
Шарапов А.Г.	3	
Рубаников М.В.		Не аттестован по причине болезни. Справка прилагается.

Последовательность действий:

1. Выполните анализ исходных документов, составляющих описание предметной области. Выделите характерные сущности.

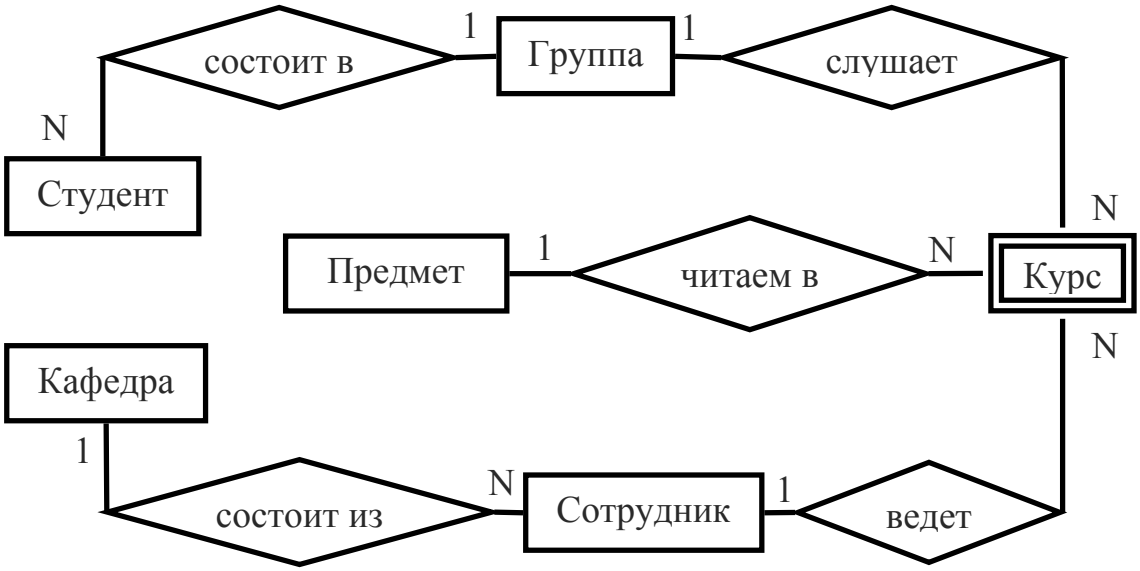
2. Выявите атрибуты для каждой сущности из предметной области. Составьте графическое изображение сущностей и их атрибутов в компактной форме. Пример графического представления сущностей приведен на рисунке.

3. Выявите связи между сущностями, определите типы связей (один к одному, один ко многим или многие ко многим). Проанализируйте свойства связей и при необходимости введите в модель дополнительные сущности или объедините существующие.

Поскольку база данных будет строиться с использованием реляционной модели, все связи вида «многие ко многим» следует преобразовать в связи «один ко многим». Связи вида «один к одному» во многих случаях можно устранить, объединив сущности.

4. Составьте диаграмму «Сущность-связь» (ER-диаграмму), представляющую связи (взаимоотношения) между различными сущностями. Если присутствуют «слабые» сущности, они должны быть выделены особо.

Пример ER-диаграммы приведен ниже. Диаграмма относится к предметной области, которая родственна рассматриваемой.



5. Для каждой сущности определите способ уникальной идентификации экземпляров. В качестве идентификатора могут быть использованы уже существующие атрибуты, если для них соблюдено требование уникальности и их зна-

чения не подвержены изменениям. Во многих случаях может оказаться более предпочтительным введение нового числового атрибута в качестве идентификатора. Для составных сущностей уникальный идентификатор должен быть составным и включать в себя идентификаторы сущностей, которые связаны с данной.

Внесите дополнения в графическое изображение сущностей и их атрибутов, обозначив идентификаторы сущностей. Пример графического представления сущностей с идентификаторами приведен ниже.

6. Определите домены для атрибутов сущностей. Если домен некоторого атрибута достаточно специфичен, может потребоваться образование новой сущности для хранения значений данного домена.
В примере на рисунке ниже показано, как домены атрибутов «Степень» и «Звание» порождают новые сущности для хранения значений из этих доменов.

7. На основе полученной модели «Сущность-связь» составьте реляционную модель, описывающую таблицы и их первичные ключи, обеспечивающие уникальность строк в таблицах. Определите внешние ключи, обеспечивающие связи между таблицами.

8. Выполните нормализацию таблиц. Проверьте структуру каждой таблицы на соответствие первой, второй и третьей нормальным формам. Устраните несоответствия.

9. Запустите Access, создайте базу данных с именем «Школы».

10. Реализуйте созданную ранее реляционную модель, создав соответствующие таблицы. Создайте все необходимые связи между первичными и внеш-

ними ключами таблиц и определите для связей свойства обеспечения целостности данных.

11. В тех таблицах, где это необходимо, определите свойства подстановки значений.

12. Внесите данные в таблицы. Должно быть представлено не менее 3 школ, в каждой школе не менее 3 классов и в классе не менее 4 учеников.

14. Создайте объекты-запросы, формирующие наборы данных, соответствующие исходным документам, использованным при проектировании (список школ, список классов, список учеников, список предметов, предметы, преподаваемые в указанном классе, ведомость класса по предмету). Запросы могут содержать параметры.

15. Создайте отчет «Сводная ведомость школы», выводящий список учеников всех классов школы с их годовыми оценками по всем предметам. Список группируется по названиям предметов и номерам классов. Для удобства реализации данного отчета может понадобиться создать запрос, формирующий набор всех необходимых данных.

Литература

1. Бобцов А.А., Шиегин В.В. Банки и базы данных. Основы работы с MS Access. Часть 1 (для пользователей). Учебное пособие. – СПб., 2005.
2. Карпов Б. Microsoft Access 2000: Справочник // СПб: «Питер», 2001.
3. Форт С., Хоуи Т., Релстон Дж. Программирование в среде Access 2000. Энциклопедия пользователя // Киев: Издательство «ДиаСофт», 2000.
4. Справка по Microsoft Access (входит в состав пакета Access).
5. Кириллов В.В., Громов Г.Ю. Структуризированный язык запросов (SQL). Учебное пособие // СПбГИТМО(ТУ). (электронная версия доступна в интернет по адресу http://citforum.ru/database/sql_kg/)
6. Кузнецов С.Д. Введение в стандарты языка баз данных SQL // в электронной форме: http://citforum.ru/database/sqlbook/
7. Мартин Грабер. Понимание SQL (Understanding SQL) // в электронной форме: http://www.sql.ru/docs/sql/u_sql/
8. Кириллов В.В. Основы проектирования реляционных баз данных. Учебное пособие // СПбГИТМО(ТУ). (электронная версия доступна в интернет по адресу http://citforum.ru/database/dbguide/)
9. Джен Л. Харрингтон. Проектирование реляционных баз данных // М.: «Лори», 2000.
10. Ролланд Ф.Д. Основные концепции баз данных // М.- СПб - Киев: «Вильямс», 2002.
11. Дейт К. Введение в системы баз данных, 6 изд. // Киев: «Диалектика», 1998.
12. Крёнке Д. Теория и практика построения баз данных, 9 изд. // СПб: «Питер», 2005.
13. Пушников А.Ю. Введение в системы управления базами данных (в 2 ч.): Учебное пособие // Изд-е Башкирского ун-та. - Уфа, 1999. (электронная версия доступна в интернет по адресу http://citforum.ru/database/dblearn/)

Кафедра КОТ

История кафедры КОТ неразрывно связана с развитием университета в последнее десятилетие, с превращением ИТМО в ведущий компьютерный вуз страны. Научную и научно-методическую основу создания и развития кафедры определили два события в жизни университета:

1999 год: создание центра дистанционного обучения (ЦДО) университета.

2000 год: открытие Санкт-Петербургского регионального центра «Федерации Интернет Образования» (СПб РЦ ФИО).

ЦДО ИТМО был создан как полигон для разработки дистанционных курсов университета. Сегодня ЦДО стал не только научно-методическим центром ИТМО, но и реальным участником учебного процесса вуза. Аттестации студентов, виртуальные лаборатории в среде системы ДО ИТМО стали неотъемлемой частью учебного процесса различных кафедр университета, разумеется, и кафедры КОТ.

СПб РЦ ФИО создан в университете в рамках негосударственного образовательного проекта «Поколение.ru» для массовой переподготовки работников среднего образования в области интернет-технологий. Успешная работа в этом направлении позволила обучить к сентябрю 2005 года более 6000 учителей. Методики и программы обучения интернет-технологиям на курсах СПб РЦ ФИО стали основой многих студенческих учебных программ.

2001 год: открытие на факультете ИТП кафедры КОТ.

Вначале кафедра КОТ обеспечивала учебный процесс по специальности 230201 (старый индекс 071900) – «Информационные системы и технологии». Дисциплины кафедры: «Информатика», «Информационные технологии», «Web-программирование».

2003 год:

- открытие в ИТМО новой инженерной специальности 230202 (073700) – «Информационные технологии в образовании»;
- переход кафедры КОТ в разряд выпускающих кафедр;
- разработан и утвержден учебный план по специализации «Аппаратно-программные комплексы образовательных систем»;
- произведен первый набор студентов специальности 230202.

С этого времени развитие кафедры подчинено главной цели: качественной подготовке специалистов высшей квалификации в области информационных технологий для сферы образования.

Для заметок

Для заметок

Алексей Алексеевич Бобцов

Виталий Валерьевич Шиегин

Банки и базы данных. Основы работы с MS Access.
Часть 2 (для разработчиков)

Учебное пособие

В авторской редакции
компьютерный набор и верстка А.А. Бобцов
Редакционно-издательский отдел СПбГУ ИТМО
Зав. РИО Н.Ф. Гусарова
Лицензия ИД № 00408 от 05.11.99
Подписано к печати
Тираж 100 экз. Заказ № 851
Отпечатано на ризографе.