

# Database Theory



# Урок № 4

## Многотабличные базы данных

## Содержание

<b>1. Аномалии взаимодействия с однотабличной базой данных .....</b>	<b>4</b>
Аномалии обновления .....	5
Аномалии вставки .....	6
Аномалии удаления .....	7
<b>2. Принципы создания многотабличной базы данных .....</b>	<b>8</b>
Причины создания многотабличной базы данных .....	8
Внешний ключ .....	8
Связи. Типы связей .....	10
Целостность данных .....	17
Нормализация .....	18

Необходимость нормализации . . . . .	18
Понятие нормальной формы . . . . .	19
Первая нормальная форма . . . . .	20
Вторая нормальная форма . . . . .	22
Третья нормальная форма . . . . .	24
Нормальная форма Бойса-Кодда . . . . .	26
<b>3. Многотабличные запросы . . . . .</b>	<b>29</b>
Принципы создания многотабличного запроса . . . . .	29
Декартовое произведение . . . . .	34
<b>4. Домашнее задание . . . . .</b>	<b>36</b>

# 1. Аномалии взаимодействия с однотабличной базой данных

При использовании однотабличных баз данных имеют место три специфические проблемы, связанные с обновлением, удалением и вставкой данных. Осуществление указанных операций с таблицами базы данных может привести к противоречивости хранящихся в них данных, что в целом отрицательно скажется на работе с этой базой данных. Такого рода операции называются аномалиями, то есть тем, что является отклонением от нормы.

Для того чтобы продемонстрировать вам проблемы, связанные с однотабличной базой данных, мы воспользуемся таблицей **Teachers**, представленной на рисунке 1.1.

Name	BirthDate	Department	Phone	Group	Subject
Sophia Nelson	1984-12-08	Software development	32-12	31PPS11	C#
Emma Kirk	1973-05-12	Mathematics	55-34	32PR31	Discrete Math
Henry MacAlister	1975-02-17	Software development	32-12	30PR11	SQL Server
Michael Cooper	1978-11-23	Software development	32-12	29PR21	ADO.NET
Daniel Williams	1979-07-30	Cybersecurity	37-65	32PPS11	ITE1
Sophia Nelson	1984-12-08	Software development	32-12	30PR11	JavaScript
Daniel Williams	1979-07-30	Cybersecurity	37-65	32PPS11	WIN10

Рисунок 1.1. Таблица Teachers

Как вы заметили, в данной таблице хранятся сведения о кафедрах, преподавателях, предметах и группах, в которых преподаются указанные предметы.

## Аномалии обновления

Аномалия обновления связана с избыточностью данных, хранимых в соответствующей таблице. Избыточность данных является причиной того, что в процессе обновления информации в таблице часть дублируемых данных не будут изменены, что приведет к противоречивости хранимой информации.

Существует два вида избыточности: явная и неявная. Явная избыточность выражается в дублировании одинаковой информации, например, в таблице **Teachers** некоторые данные о преподавателе **Sophia Nelson** повторяются несколько раз. В том случае если у нее изменится фамилия на **Davies** (она выйдет замуж), то для того чтобы избежать противоречивости данных в таблице, необходимо будет вносить изменения в каждую запись об этом преподавателе (Рисунок 1.2).

Name	BirthDate	Department	Phone	Group	Subject
<u>Sophia Davies</u>	1984-12-08	Software development	32-12	31PPS11	C#
Emma Kirk	1973-05-12	Mathematics	55-34	32PR31	Discrete Math
Henry MacAlister	1975-02-17	Software development	32-12	30PR11	SQL Server
Michael Cooper	1978-11-23	Software development	32-12	29PR21	ADO.NET
Daniel Williams	1979-07-30	Cybersecurity	37-65	32PPS11	ITE1
<u>Sophia Nelson</u>	1984-12-08	Software development	32-12	30PR11	JavaScript
Daniel Williams	1979-07-30	Cybersecurity	37-65	32PPS11	WIN10

**Рисунок 1.2.** Явная избыточность

Неявная избыточность проявляется в виде неявной зависимости между различными записями одной таблицы, например, номер телефона кафедры **Software development** повторяется в сведениях о нескольких преподавателях.

Допустим, если в случае изменения на кафедре номера телефона (на 48-22) мы внесем изменения только в запись о преподавателе **Henry MacAlister**, то при последующей работе с таблицей **Teachers** мы, фактически, не сможем определить правильный номер телефона кафедры, так как в записях о преподавателях **Sophia Nelson** и **Michael Cooper** он останется прежним (Рисунок 1.3).

Name	BirthDate	Department	Phone	Group	Subject
Sophia Nelson	1984-12-08	Software development	32-12	31PPS11	C#
Emma Kirk	1973-05-12	Mathematics	55-34	32PR31	Discrete Math
Henry MacAlister	1975-02-17	Software development	48-22	30PR11	SQL Server
Michael Cooper	1978-11-23	Software development	32-12	29PR21	ADO.NET
Daniel Williams	1979-07-30	Cybersecurity	37-65	32PPS11	ITE1
Sophia Nelson	1984-12-08	Software development	32-12	30PR11	JavaScript
Daniel Williams	1979-07-30	Cybersecurity	37-65	32PPS11	WIN10

Рисунок 1.3. Неявная избыточность

## Аномалии вставки

Аномалия вставки проявляется в тех случаях, когда существует необходимость поместить в таблицу запись, у которой отсутствует часть информации. Отсутствие некоторых данных в таблице, в дальнейшем, может послужить причиной получения неверных результатов при выполнении запросов к базе данных. Это не является критичным если поле с отсутствующими данными может иметь неопределенное значение (**NULL**-значения), однако для полей, обязательных к заполнению, такая ситуация приведет к возникновению ошибки. В нашем примере добавление в таблицу **Teachers** преподавателя, о котором неизвестны группа и читаемый им предмет невозможно,

## 1. Аномалии взаимодействия с однотабличной базой данных

поэтому мы не сможем «принять на работу» человека пока не выясним эту информацию (Рисунок 1.4).

The screenshot shows a SQL query window titled "SQLQuery1.sql - DE...ESKTOP\Yuriy (54)\*". The query is:

```
INSERT INTO Teachers ([Name], BirthDate, Department, Phone, [Group], [Subject])
VALUES ('John Doe', '1974-03-22', 'Mathematics', '55-34', NULL, NULL);
```

The "Messages" pane displays the following error:

```
Msg 515, Level 16, State 2, Line 1
Cannot insert the value NULL into column 'Group', table 'University.dbo.Teachers';
column does not allow nulls. INSERT fails.
The statement has been terminated.
```

Рисунок 1.4. Ошибка: невозможно записать NULL-значение

### Аномалии удаления

Аномалия удаления связана с уникальностью информации в определенных записях, при удалении которых эти данные будут потеряны. Например, в нашем случае если преподаватель [Emma Kirk](#) уволиться с работы, и мы удалим из таблицы [Teachers](#) запись содержащую информацию о ней, то это приведет к потере данных о предмете [Discrete Math](#), кафедре [Mathematics](#) и группе [32PR31](#), потому что информация о них содержится только в этой записи (Рисунок 1.5).

Name	BirthDate	Department	Phone	Group	Subject
Sophia Nelson	1984-12-08	Software development	32-12	31PPS11	C#
Emma Kirk	1973-05-12	Mathematics	55-34	32PR31	Discrete Math
Henry MacAlister	1975-02-17	Software development	32-12	30PR11	SQL Server
Michael Cooper	1978-11-23	Software development	32-12	29PR21	ADO.NET
Daniel Williams	1979-07-30	Cybersecurity	37-65	32PPS11	ITE1
Sophia Nelson	1984-12-08	Software development	32-12	30PR11	JavaScript
Daniel Williams	1979-07-30	Cybersecurity	37-65	32PPS11	WIN10

Рисунок 1.5. Аномалия удаления

## 2. Принципы создания многотабличной базы данных

*Многотабличная база данных* — это определенная структура для хранения различной информации, которая состоит из связанных между собой таблиц. Создание многотабличной базы данных является трудоемким процессом, но для хранения данных необходимо использовать именно такой подход, на что существует ряд причин.

### **Причины создания многотабличной базы данных**

При хранении всей информации в одной таблице необходимо будет создавать большое количество столбцов, при этом объем хранимой информации будет увеличиваться пропорциональным образом. Записи в этой таблице будут во многом дублировать данные, что также приведет к увеличению объема информации, которая храниться в базе. Все это сказывается на процессе обработки данных, увеличивая время выполнения SQL-запросов. Нельзя также забывать и об аномалиях, описанных в предыдущем разделе, которые присущи однотабличным базам данных. Все вышеперечисленное является вескими причинами для использования именно многотабличных баз данных.

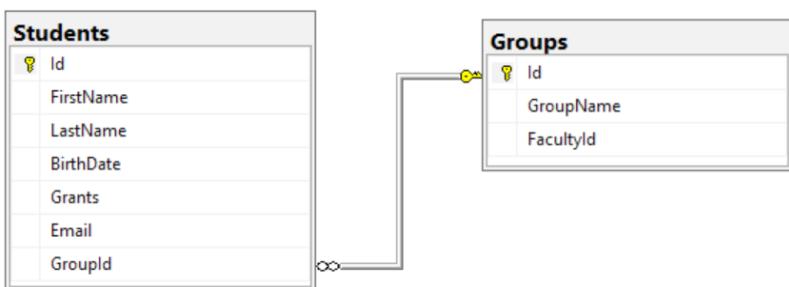
### **Внешний ключ**

Прежде чем мы начнем обсуждать типы связей между таблицами в многотабличной базе данных, вам необходимо понять сам механизм этих связей.

Из урока № 2 вы узнали, что в любой таблице должен быть столбец, который обеспечивает уникальность записей в таблице — первичный ключ. Также в таблицах может быть еще одно специализированное поле, которое называется внешним ключом.

Внешний ключ — это поле таблицы, в котором содержится значение первичного ключа другой таблицы. Именно благодаря соотношению первичного ключа одной таблицы и внешнего ключа другой и формируются связи между таблицами в многотабличной базе данных.

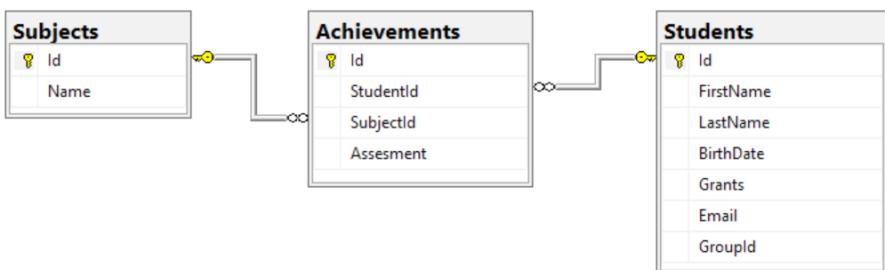
В качестве примера приведем часть диаграммы базы данных, на которой в таблицах **Students** и **Groups** первичными ключами являются столбцы **Id**, что визуально отмечено изображениями ключа (Рисунок 2.1).



**Рисунок 2.1.** Пример формирования связи между таблицами

В таблице **Students** внешним ключом является столбец **GroupId**, в котором будут храниться данные из столбца **Id** — первичного ключа таблицы **Groups**. В свою очередь столбец **FacultyId** в таблице **Groups** является внешним ключом таблицы **Faculties** (на диаграмме не изображена).

Важно отметить, что в одной таблице допускается наличие нескольких столбцов, являющихся внешними ключами, каждый из которых будет хранить значения первичных ключей различных таблиц текущей базы данных. В качестве примера приведем таблицу **Achievements**, которая содержит информацию об успеваемости студентов по различным предметам (Рисунок 2.2).



**Рисунок 2.2.** Пример таблицы с несколькими внешними ключами

В таблице **Achievements** внешними ключами являются столбцы **StudentId** и **SubjectId**, в которых будут храниться значения первичных ключей (столбцы **Id**) таблиц **Students** и **Subjects** соответственно.

## Связи. Типы связей

На рисунке 2.1 показаны таблицы и схематичное изображение связи между таблицами **Students** и **Groups**, которая создается благодаря наличию первичного ключа в таблице **Groups** (столбец **Id**) и внешнего ключа в таблице **Students** (столбец **GroupId**). Как вы, наверное, догадались, одна сторона связи с изображением ключа соединяется с таблицей, в которой находится первичный ключ, противоположная

сторона связи в виде значка бесконечности соединяется с таблицей, содержащей внешний ключ.

При проектировании любой реляционной базы данных используются три типа связей: «один к одному», «один ко многим» и «многие ко многим».

Связь «один к одному» существует между двумя таблицами в том случае, если строка данных в первой таблице соответствует только одной строке второй таблицы, а строка во второй таблице связана только с одной строкой данных первой таблицы. Этот тип связи графически представляется в виде линии, соединяющей две таблицы. На рисунке 2.3 приведен пример связи такого типа, между таблицами *Teachers* и *Authentications*.

Teachers				Authentications		
<u>Id</u>	LastName	FirstName	BirthDate	<u>Id</u>	Login	Password
1	Nelson	Sophia	1984-12-08			
2	Kirk	Emma	1973-05-12			
3	MacAlister	Henry	1975-02-17			
4	Cooper	Michael	1978-11-23			
5	Williams	Daniel	1979-07-30			

**Рисунок 2.3.** Пример связи «один к одному»

В обеих таблицах существует первичный ключ (столбец *Id*), значения в этих столбцах должны быть одинаковыми для соответствующих записей в таблицах — у любого преподавателя может быть только одно сочетание логина и пароля для входа в систему выставления оценок — тем самым формируя связь «один к одному».

Связь такого типа используется в реляционных базах данных очень редко. Данный тип связи может быть реализован путем разделения одной таблицы на две, в том

случае если в таблице содержится большое количество столбцов, и информация по некоторым из них требуется крайне редко. На рисунке 2.3 продемонстрирована еще одна ситуация, которая может возникнуть, когда часть данных в таблице носит секретный характер, в этом случае секретные данные выносятся в отдельную таблицу, которой устанавливается более высокий уровень безопасности (будет рассмотрено в курсе MS SQL Server), тем самым ограничивая доступ к этой информации.

Связь «один ко многим» является наиболее часто используемым типом отношений и реализуется между двумя таблицами, когда строке в первой таблице соответствует множество строк во второй таблице, но строка второй таблицы должна быть связана только с одной строкой данных первой таблицы. Этот тип связи графически представляется в виде линии, один конец которой оканчивается трезубцем, называемым также «вороньей лапкой». На рисунке 2.4 приведен пример связи такого типа, между таблицами **Students** и **Groups**.

Students							Groups		
<b><u>Id</u></b>	<b><u>FirstName</u></b>	<b><u>LastName</u></b>	<b><u>BirthDate</u></b>	<b><u>Grants</u></b>	<b><u>Email</u></b>	<b><u>GroupId</u></b>	<b><u>Id</u></b>	<b><u>GroupName</u></b>	<b><u>FacultyId</u></b>
1	Jack	Jones	1997-11-05	1256.00	jj@net.eu	1	1	29PR21	1
2	Harry	Miller	1998-02-11	1100.00	hm@net.eu	1	2	30PR11	2
3	Grace	Evans	1997-06-24	NULL	eg@net.eu	2	3	31PPS11	1
4	Lily	Wilson	1998-09-12	NULL	lw@net.eu	2	4	32PR31	2
5	Joshua	Johnson	1997-05-23	1100.00	jo@net.eu	3	5	32PPS11	3
6	Emily	Taylor	1997-12-27	1100.00	et@net.eu	4			
7	Charlie	Thomas	1998-01-31	1256.00	ct@net.eu	4			
8	Oliver	Moore	1997-07-05	NULL	om@net.eu	4			
9	Jessica	Brown	1997-07-17	1100.00	jb@net.eu	5			

Рисунок 2.4. Пример связи «один ко многим»

Действительно в любой группе может учиться определенное количество студентов (множество), но каждый

конкретный студент находится только в одной группе. Например, в нашем случае в группе 32PR31 учатся несколько студентов: Emily Taylor, Charlie Thomas и Oliver Moore.

Связь «многие ко многим» может существовать между двумя таблицами реляционной базы данных в том случае, если строке первой таблицы соответствует множество строк со второй таблицы и строке во второй таблицы соответствует множество строк из первой таблицы. Этот тип связи графически представляется в виде пунктирной линии, оба конца которой оканчиваются трезубцем. Например, преподаватель может проводить занятия в нескольких группах и в одной группе могут преподавать определенное количество преподавателей. На рисунке 2.5 приведен пример связи такого типа, между таблицами Teachers и Groups.

The diagram shows two tables, 'Teachers' and 'Groups', connected by a dashed arrow pointing from 'Groups' to 'Teachers'. The 'Groups' table has columns: ID (1-5), Name (Группа 1-5). The 'Teachers' table has columns: ID (1-6), Name (Имя преподавателя 1-6), and Date of birth (Дата рождения).

ID	Name	Date of birth
1	Emily Taylor	1998-01-30
2	Charlie Thomas	1997-11-15
3	Henry Moore	1992-05-11
4	Kirk	1993-02-15
5	Oliver Moore	1995-01-08
6	John Doe	1996-07-20

ID	Name	Group
1	Группа 1	
2	Группа 2	
3	Группа 3	
4	Группа 4	
5	Группа 5	

Рисунок 2.5. Пример связи «многие ко многим»

Данный тип связи является логическим и не может быть физически реализован в таком виде в базе данных. Поэтому для организации такого типа связи в базе данных используется следующий подход: между двумя таблицами, соединенными связью «многие ко многим», добавляется третья таблица, в которой находятся внешние

ключи для первой и второй таблицы, тем самым устанавливается связь «один ко многим» с каждой из этих таблиц. На основании вышеизложенного пример связи на рисунке 2.5 может быть реализован в следующем виде (Рисунок 2.6).

Teachers			
<b>Id</b>	<b>LastName</b>	<b>FirstName</b>	<b>BirthDate</b>
1	Nelson	Sophia	1984-12-08
2	Kirk	Emma	1973-05-12
3	MacAlister	Henry	1975-02-17
4	Cooper	Michael	1978-11-23
5	Williams	Daniel	1979-07-30

TeachersGroups	
<b>TeacherId</b>	<b>GroupId</b>
1	3
2	4
3	2
4	1
5	5
1	2

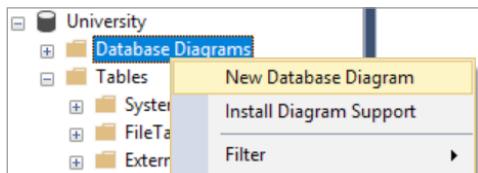
Groups		
<b>Id</b>	<b>GroupName</b>	<b>FacultyId</b>
1	29PR21	1
2	30PR11	2
3	31PPS11	1
4	32PR31	2
5	32PPS11	3

**Рисунок 2.6.** Реализация связи «многие ко многим»

В таблице **TeachersGroups** столбцы **TeacherId** и **GroupId** являются внешними ключами для первичных ключей (столбцы **Id**) таблиц **Teachers** и **Groups**, соответственно, благодаря чему формируются связи «один ко многим» между таблицами **TeachersGroups** и **Teachers**, и **TeachersGroups** и **Groups**.

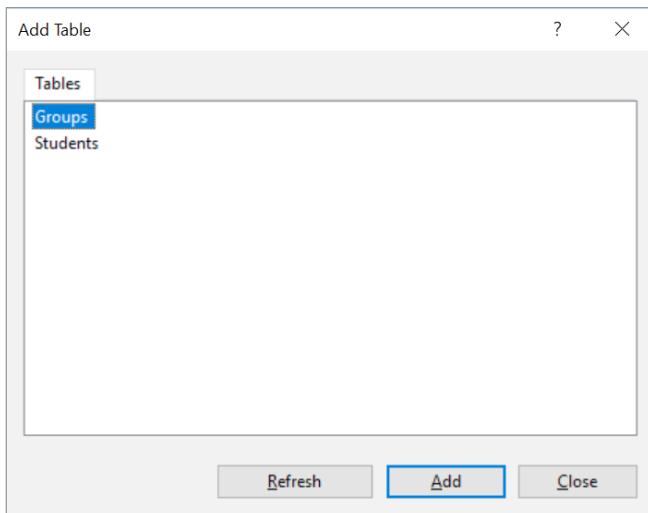
После того как вы получили информацию о связях между таблицами было бы неплохо продемонстрировать вам как они формируются при помощи MS SQL Server Management Studio 17. Именно это мы сейчас и рассмотрим на примере связи «один ко многим» (как наиболее часто применяемой), которая показана на рисунке 2.4.

Для того чтобы сформировать связи между таблицами необходимо создать диаграмму базы данных, вызвав контекстное меню на пункте **Database Diagrams** в разделе **Databases** окна **Object Explorer** у требуемой базы данных (**University**), и выбрать в нем пункт **New Database Diagram** (Рисунок 2.7).



**Рисунок 2.7.** Создание диаграммы базы данных (начало)

После этого появиться форма **Add Table** со списком всех таблиц, которые существуют в текущей базе данных. Для того чтобы добавить их на диаграмму необходимо выбрать в списке нужные таблицы и нажать кнопку **Add** (Рисунок 2.8).



**Рисунок 2.8.** Добавление необходимых таблиц

В результате выполнения этих действий вы получите диаграмму требуемой базы данных. Нам осталось установить связи между таблицами, для того чтобы это сделать необходимо кликнуть левой клавишей мыши по

изображению ключа в поле первичного ключа **Id** таблицы **Groups** и не отпуская клавиши протянуть мышь на поле внешнего ключа **GroupId** таблицы **Students** и отпустить клавишу (Рисунок 2.9).

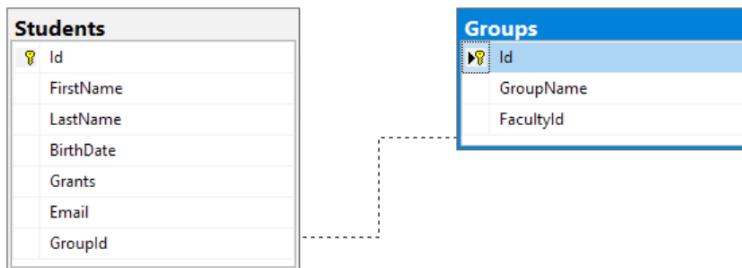


Рисунок 2.9. Создание связи между таблицами

Сразу же после этого вы увидите две формы **Tables and Columns** и **Foreign Key Relationship** (Рисунок 2.10).

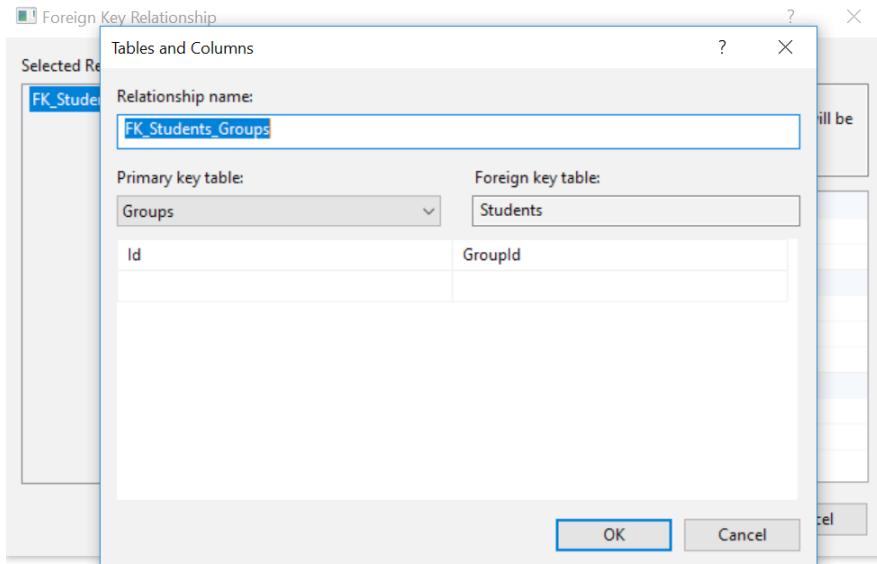


Рисунок 2.10. Формирование связи между таблицами

На форме **Tables and Columns** выводится информация о первичном и внешнем ключе и название отношения между ними, в случае необходимости вы можете подкорректировать любые данные, если вас все устраивает, тогда вы нажимаете кнопку OK. Следующая форма **Foreign Key Relationship** позволяет более точно настроить само отношение, для того чтобы закончить формирования связи между таблицами необходимо нажать кнопку OK. После этого отношение между таблицами будет сохранено в текущей базе данных, и вы увидите его на диаграмме базы данных, которая представлена на рисунке 2.1.

## Целостность данных

При работе с различными базами данных существует вероятность записи в таблицы ошибочной информации, которая приведет к нарушению целостности данных и как результат к некорректной работе всей базы данных.

Язык T-SQL, как язык реляционной СУБД, поддерживает проверку входных данных и обеспечивает их целостность благодаря наличию ряда ограничений:

- **NOT NULL** — ограничение, которое гарантирует, что в текущий столбец каждой записи обязательно будут записаны данные, иначе попытка добавления записи в таблицу без этого значения приведет к ошибке;
- **DEFAULT** — ограничение, которое обеспечивает запись в столбец значения заданного по умолчанию в случае, если не указана другая информация;
- **CHECK** — ограничение, которое задает условие проверки вводимых данных, если условие не выполняется, то запись в таблицу не добавляется;

- **UNIQUE** — ограничение, которое обеспечивает уникальность записанной информации в соответствующем столбце;
- **PRIMARY KEY** — ограничение, которое указывает что текущий столбец (столбцы) является первичным ключом данной таблицы, данное ограничение может быть только одно в таблице и является комбинацией ограничений **UNIQUE** и **NOT NULL**;
- **FOREIGN KEY** — ограничение, которое указывает что текущий столбец (столбцы) является внешним ключом, данное ограничение обеспечивает ссылочную целостность данных.

На самом деле вам не обязательно указывать ограничения для столбцов таблицы, потому что целостность данных можно обеспечить и другими способами, например, при помощи триггеров (будут рассмотрены в курсе MS SQL Server) или на уровне приложения. Однако этот способ защиты данных является самым лучшим, потому что проверка ограничений выполняется сервером в первую очередь и естественно происходит быстрее всех других проверок.

## **Нормализация**

Одной из самых важных концепций в реляционной базе данных является концепция нормализованных данных.

### **Необходимость нормализации**

*Нормализованные данные* — это данные, организованные в такую структуру, которая обеспечивает целостность информации, хранящейся в базе данных,

и сводит к минимуму количество избыточных данных. Структурирование информации в базе данных делается не только с целью экономии памяти, но и для предотвращения появления несогласованных данных, связанных с различными аномалиями, а также для исключения потери информации в процессе эксплуатации базы данных.

Нормализация данных осуществляется путем разбиения (декомпозиции) одной таблицы на две или более. При этом формируются необходимые связи между полученными таблицами с целью сохранения согласованности между данными.

### ***Понятие нормальной формы***

Процесс нормализации данных выполняется в соответствии с общепринятыми правилами нормализации, которые регламентируют порядок получения нормализованной базы данных и называются **нормальными формами**.

Впервые понятие нормальные формы было введено Эдгаром Коддом при описании им реляционной модели базы данных. Изначально было предложено три нормальные формы, но прогресс не стоит на месте и на текущий момент времени количество нормальных форм достигло уже восьми:

- первая нормальная форма (1NF);
- вторая нормальная форма (2NF);
- третья нормальная форма (3NF);
- нормальная форма Бойса-Кодда(BCNF);
- четвертая нормальная форма (4NF);
- пятая нормальная форма (5NF);

- доменно-ключевая нормальная форма (DKNF);
- шестая нормальная форма (6NF).

Перечисленные выше нормальные формы не являются обязательными для выполнения при проектировании базы данных, они скорее являются рекомендациями, которые вы вправе реализовать частично. При этом вам следует знать, что нормальные формы связаны между собой неразрывной последовательностью, в которой каждая последующая нормальная форма основана на предыдущей.

Существенным показателем качества спроектированной базы данных является ее производительность, которая напрямую связана со степенью нормализации данных — чем более высокой нормальной форме соответствует база данных, тем больше ресурсов системы необходимо для ее сопровождения. Поэтому спроектированная вами база данных должна соблюдать баланс между степенью нормализации данных и производительностью системы. Обычно чтобы достигнуть этой цели достаточно реализовать в базе данных первые четыре нормальные формы из приведенного выше списка.

### ***Первая нормальная форма***

Первая нормальная форма является наиболее важной и служит основой для всех остальных нормальных форм. Для того чтобы таблица соответствовала первой нормальной форме она должна отвечать следующим требованиям:

- каждое значение в записи должно быть атомарным (неделимым), то есть любой столбец в таблице должен содержать только одно значение для каждой строки;

- все записи в таблице должны быть разными, даже если в нескольких записях содержится одинаковая информация, то вся запись в целом должна быть уникальной для таблицы.

Последнее требование соответствует наличию в каждой таблице уникального поля — первичный ключ.

Таблица `Teachers` на рисунке 1.1 не соответствует требованиям первой нормальной формы, так как записи в столбце `Name` не являются атомарными, потому что содержат как имя, так и фамилию преподавателя, также в этой таблице отсутствует первичный ключ, что не гарантирует уникальности каждой записи (Рисунок 2.11).

Name	BirthDate	Department	Phone	Group	Subject
Sophia Nelson	1984-12-08	Software development	32-12	31PPS11	C#
Emma Kirk	1973-05-12	Mathematics	55-34	32PR31	Discrete Math
Henry MacAlister	1975-02-17	Software development	32-12	30PR11	SQL Server
Michael Cooper	1978-11-23	Software development	32-12	29PR21	ADO.NET
Daniel Williams	1979-07-30	Cybersecurity	37-65	32PPS11	ITE1
Sophia Nelson	1984-12-08	Software development	32-12	30PR11	JavaScript
Daniel Williams	1979-07-30	Cybersecurity	37-65	32PPS11	WIN10

Рисунок 2.11. Несоответствие таблицы  
первой нормальной форме

Id	LastName	FirstName	BirthDate	Department	Phone	Group	Subject
1	Nelson	Sophia	1984-12-08	Software development	32-12	31PPS11	C#
2	Kirk	Emma	1973-05-12	Mathematics	55-34	32PR31	Discrete Math
3	MacAlister	Henry	1975-02-17	Software development	32-12	30PR11	SQL Server
4	Cooper	Michael	1978-11-23	Software development	32-12	29PR21	ADO.NET
5	Williams	Daniel	1979-07-30	Cybersecurity	37-65	32PPS11	ITE1
6	Nelson	Sophia	1984-12-08	Software development	32-12	30PR11	JavaScript
7	Williams	Daniel	1979-07-30	Cybersecurity	37-65	32PPS11	WIN10

Рисунок 2.12. Приведение таблицы  
к первой нормальной форме

Для того чтобы привести таблицу **Teachers** в соответствие с первой нормальной формой необходимо внести ряд изменений: добавить первичный ключ (столбец **Id**) и создать отдельные столбцы для имени и фамилии (Рисунок 2.12).

### **Вторая нормальная форма**

Прежде чем дать определение второй нормальной формы необходимо ввести ряд понятий, ключевым из которых является понятие функциональной зависимости.

**Функциональная зависимость** определяется следующим образом: при существовании двух полей **X** и **Y**, поле **X** функционально зависит от **Y**, если с любым значением поля **X** всегда связано ровно одно значение поля **Y**. Значения полей **X** и **Y** могут изменяться в течение времени, но только таким образом, чтобы любое поле **X** имело, связанное с ним уникальное значение **Y**. Обозначается функциональная зависимость следующим образом:  $X \rightarrow Y$ . В нашем случае можно привести следующие примеры функциональной зависимости: **LastName**  $\rightarrow$  **FirstName**, **LastName**  $\rightarrow$  **Department** и т.д.

Следующее понятие **полной функциональной зависимости** можно сформулировать следующим образом: если ключевое поле **X** состоит из нескольких полей, то неключевое поле **Y** должно функционально полно зависеть от **X** в целом и не зависеть функционально от какого-либо поля, входящего в него. Например, если в таблице **Teachers** первичный ключ состоит из столбцов **LastName** и **FirstName**, то столбец **BirthDate** будет находиться в полной функциональной зависимости от первичного ключа.

**Частичной функциональной зависимостью**, соответственно, называется зависимость неключевого поля от части составного ключевого поля.

**Неключевое поле** — это столбец, который не входит в состав ни одного из возможных потенциальных ключей.

Требования второй нормальной формы расширяют первую нормальную форму, беря ее за основу:

- таблица должна выполнять условия первой нормальной формы;
- все столбцы, которые не входят в первичный ключ, должны зависеть от первичного ключа в целом, то есть должны быть связаны полной функциональной зависимостью с первичным ключом.

Последнее требование соответствует отсутствию в таблице частичных функциональных зависимостей.

Если предположить, что в таблице **Teachers** первичный ключ состоит из столбцов **Last Name**, **Group** и **Subject**, тогда столбцы **First Name**, **Birth Date**, **Department** и **Phone** образуют частичную функциональную зависимость, так как они зависят от столбца **Last Name** и не зависят от столбцов **Group** и **Subject** (Рисунок 2.13).

<b>Id</b>	<b>Last Name</b>	<b>First Name</b>	<b>Birth Date</b>	<b>Department</b>	<b>Phone</b>	<b>Group</b>	<b>Subject</b>
1	Nelson	Sophia	1984-12-08	Software development	32-12	31PPS11	C#
2	Kirk	Emma	1973-05-12	Mathematics	55-34	32PR31	Discrete Math
3	MacAlister	Henry	1975-02-17	Software development	32-12	30PR11	SQL Server
4	Cooper	Michael	1978-11-23	Software development	32-12	29PR21	ADO.NET
5	Williams	Daniel	1979-07-30	Cybersecurity	37-65	32PPS11	ITE1
6	Nelson	Sophia	1984-12-08	Software development	32-12	30PR11	JavaScript
7	Williams	Daniel	1979-07-30	Cybersecurity	37-65	32PPS11	WIN10

**Рисунок 2.13.** Несоответствие таблицы второй нормальной форме

Данная проблема решается путем разбиения существующей таблицы **Teachers** на две таблицы следующим образом:

- в первой таблице (**Teachers**) будут содержаться все столбцы, которые находятся в функциональной зависимости от части первичного ключа вместе с этой частью;
- во второй таблице (**GroupsSubjects**) будут находиться остальные части первичного ключа и зависящие от них столбцы (Рисунок 2.14).

Teachers						GroupsSubjects		
<b>Id</b>	<b>LastName</b>	<b>FirstName</b>	<b>BirthDate</b>	<b>Department</b>	<b>Phone</b>	<b>Id</b>	<b>Group</b>	<b>Subject</b>
1	Nelson	Sophia	1984-12-08	Software development	32-12	1	31PPS11	C#
2	Kirk	Emma	1973-05-12	Mathematics	55-34	2	32PR21	Discrete Math
3	MacAlister	Henry	1975-02-17	Software development	32-12	3	30PR11	SQL Server
4	Cooper	Michael	1978-11-23	Software development	32-12	4	29PR21	ADO.NET
5	Williams	Daniel	1979-07-30	Cybersecurity	37-65	5	32PPS11	ITE1
6	Nelson	Sophia	1984-12-08	Software development	32-12	6	30PR11	JavaScript
7	Williams	Daniel	1979-07-30	Cybersecurity	37-65	7	32PPS11	WIN10

**Рисунок 2.14.** Приведение таблиц  
ко второй нормальной форме

### Третья нормальная форма

С понятием третьей нормальной формы напрямую связано понятие транзитивной функциональной зависимости.

Допустим, в таблице существуют три поля X, Y и Z, тогда функциональная зависимость между полями X и Z ( $X \rightarrow Z$ ) называется **транзитивной**, если существуют зависимости между полями X и Y ( $X \rightarrow Y$ ) и полями Y и Z ( $Y \rightarrow Z$ ).

Требования третьей нормальной формы основаны на второй нормальной форме и звучат следующим образом:

- таблица должна соответствовать требованиям второй нормальной формы;
- все столбцы, не входящие в первичный ключ, должны функционально зависеть только от первичного ключа и не зависеть от любого другого неключевого столбца.

Последнее требование соответствует отсутствию транзитивных зависимостей в таблице. Например, в таблице `Teachers` присутствует транзитивная зависимость между столбцами `Id` и `Phone`, так как существуют функциональные зависимости `Id -> Department` и `Department -> Phone`, потому что номер телефона зависит от кафедры (Рисунок 2.15).

<code>Id</code>	<code>LastName</code>	<code>FirstName</code>	<code>BirthDate</code>	<code>Department</code>	<code>Phone</code>
1	Nelson	Sophia	1984-12-08	Software development	32-12
2	Kirk	Emma	1973-05-12	Mathematics	55-34
3	MacAlister	Henry	1975-02-17	Software development	32-12
4	Cooper	Michael	1978-11-23	Software development	32-12
5	Williams	Daniel	1979-07-30	Cybersecurity	37-65
6	Nelson	Sophia	1984-12-08	Software development	32-12
7	Williams	Daniel	1979-07-30	Cybersecurity	37-65

**Рисунок 2.15** Несоответствие таблицы третьей нормальной форме

Для того чтобы таблица `Teachers` соответствовала третьей нормальной форме необходимо осуществить ее разбиение на две таблицы, вынеся столбцы, которые являются причиной транзитивной зависимости, в отдельную таблицу (`Departments`) (Рисунок 2.16).

Teachers				Departments		
Id	LastName	FirstName	BirthDate	Id	Department	Phone
1	Nelson	Sophia	1984-12-08	1	Software development	32-12
2	Kirk	Emma	1973-05-12	2	Mathematics	55-34
3	MacAlister	Henry	1975-02-17	3	Cybersecurity	37-65
4	Cooper	Michael	1978-11-23			
5	Williams	Daniel	1979-07-30			

Рисунок 2.16. Приведение таблиц к третьей нормальной форме

### Нормальная форма Бойса-Кодда

Нормальная форма Бойса-Кодда иногда называют усиленной третьей нормальной формой, потому что она накладывает большие ограничения на функциональные зависимости между столбцами и применяется при сочетании нескольких условий: в таблице должно быть два или более составных потенциальных ключа при этом они должны перекрываться, то есть иметь хотя бы один общий столбец. При невыполнении указанных выше условий нормальная форма Бойса-Кодда эквивалентна третьей нормальной форме.

Требования нормальной формы Бойса-Кодда берут за основу требования третьей нормальной формы, расширяя их:

- таблица должна соответствовать требованиям третьей нормальной формы;
- столбцы, входящие в первичный ключ, не должны иметь функциональных зависимостей с любым из неключевых столбцов.

Из последнего требования следует, что в таблице должны отсутствовать перекрывающиеся потенциальные

первичные ключи. В качестве примера рассмотрим таблицу **Achievements** (Рисунок 2.17).

StudentId	RecordBookNumber	Subject	Assesment
1	07-15-18	ADO.NET	11
2	12-23-56	C#	10
1	07-15-18	SQL Server	10
3	67-34-61	C#	8
2	12-23-56	ADO.NET	9
3	67-34-61	SQL Server	7

**Рисунок 2.17.** Таблица Achievements

Данная таблица соответствует второй и третьей нормальным формам, так как в ней отсутствуют частичные и транзитивные зависимости. Однако таблица **Achievements** не соответствует нормальной форме Бойса-Кодда потому что в ней существуют два потенциальных составных ключа. Первый состоит из столбцов **StudentId** (*идентификатор студента*) и **RecordBookNumber** (*номер зачетки*), а второй из столбцов **StudentId** и **Subject**, при этом эти потенциальные ключи перекрываются, потому что столбец **StudentId** является частью обоих ключей (Рисунок 2.18).

StudentId	RecordBookNumber	Subject	Assesment
1	07-15-18	ADO.NET	11
2	12-23-56	C#	10
1	07-15-18	SQL Server	10
3	67-34-61	C#	8
2	12-23-56	ADO.NET	9
3	67-34-61	SQL Server	7

**Рисунок 2.18.** Несоответствие таблицы нормальной форме Бойса-Кодда

Для того чтобы привести таблицу Achievements в соответствие с нормальной формой Бойса-Кодда необходимо на ее основе создать две таблицы RecordBooks и Assesments, в каждой из которых столбец StudentId будет первичным ключом (Рисунок 2.19).

RecordBooks		Assesments		
StudentId	RecordBookNumber	StudentId	Subject	Assesment
1	07-15-18	1	ADO.NET	11
2	12-23-56	2	C#	10
1	07-15-18	1	SQL Server	10
3	67-34-61	3	C#	8
2	12-23-56	2	ADO.NET	9
3	67-34-61	3	SQL Server	7

**Рисунок 2.19.** Реализация нормальной формы Бойса-Кодда

После того как вы привели таблицы вашей базы данных к требуемой нормальной форме, необходимо установить связи между таблицами, которые уже были описаны в текущем разделе ранее.

# 3. Многотабличные запросы

Мы надеемся, что к текущему времени вы осознали всю важность проектирования именно многотабличных баз данных. Однако одновременно с этим приходит необходимость получения данных из различных таблиц, то есть написания многотабличных запросов, которые в простонародье называют «сложными».

## Принципы создания многотабличного запроса

Многотабличные запросы позволяют получить различную информацию из взаимосвязанных таблиц, результаты при этом, как обычно, будут отображаться в виртуальной таблице.

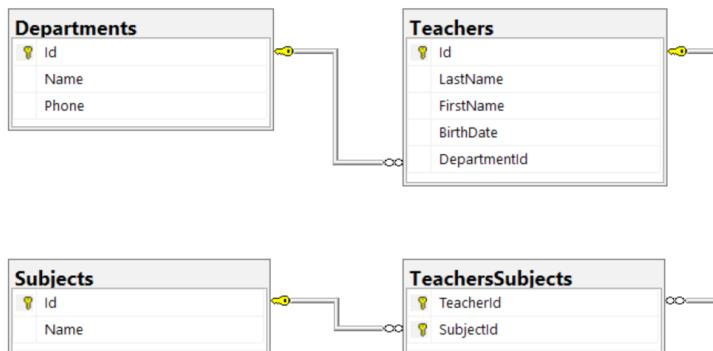


Рисунок 3.1 Диаграмма базы данных

При написании многотабличных запросов вы можете использовать все операторы языка SQL, рассмотренные нами в уроке № 3, однако существует несколько особенностей, которые необходимо всегда учитывать.

Для демонстрации написания многотабличных запросов мы воспользуемся некоторыми таблицами из тех, которые мы рассматривали в предыдущем разделе, установив между ними необходимые связи. Для того чтобы вы лучше ориентировались в этих связях, на рисунке 3.1 приведена диаграмма полученной базы данных.

На приведенной диаграмме между таблицами `Departments` и `Teachers` существует связь «один ко многим» (на кафедре может быть множество преподавателей, но отдельно взятый преподаватель может работать только на одной кафедре), которая формируется благодаря наличию внешнего ключа `DepartmentId` в таблице `Teachers`. Между таблицами `Teachers` и `Subjects` существует связь «многие ко многим» (один преподаватель может читать множество предметов и один предмет могут вести множество преподавателей), поэтому была создана третья таблица `TeachersSubjects`, в которой находятся внешние ключи `TeacherId` и `SubjectId`, которые устанавливают связь «один ко многим» с таблицами `Teachers` и `Subjects` соответственно. Как вы, наверное, заметили, в таблице `TeachersSubjects` оба столбца формируют составной первичный ключ, тем самым обеспечивается уникальность каждой записи в этой таблице.

Для того чтобы получить информацию о принадлежности преподавателей к кафедрам можно было написать SQL-запрос к одной таблице `Teachers`, но в этом случае мы получим только идентификаторы кафедр, что является не особо информативным. Поэтому необходимо написать SQL-запрос, при помощи которого мы будем получать информацию из двух таблиц `Departments` и `Teachers`:

```
SELECT FirstName + ' ' + LastName AS FullName, Name
FROM Departments, Teachers
WHERE Teachers.DepartmentId = Departments.Id;
```

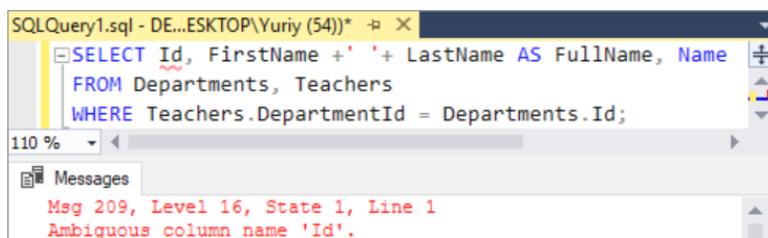
Результат, полученный после выполнения этого SQL-запроса, отображен на рисунке 3.2.

FullName	Name
Sophia Nelson	Software development
Emma Kirk	Mathematics
Henry MacAlister	Software development
Michael Cooper	Software development
Daniel Williams	Cybersecurity

Рисунок 3.2. Многотабличный запрос

Как вы заметили, для того чтобы написать многотабличный запрос необходимо соединить требуемые таблицы, указав столбцы, которые формируют соответствующую связь, в операторе **WHERE**.

Допустим, в следующем запросе мы помимо имеющейся информации захотим получить еще и идентификатор кафедры, но, если мы попытаемся в SQL-запросе указать столбец, название которого присутствует в обеих таблицах, то выполнение этого запроса приведет к ошибке (Рисунок 3.3).



The screenshot shows a SQL query window titled "SQLQuery1.sql - DE...ESKTOP\Vyuri (54)\*". The query is:

```
SELECT Id, FirstName + ' ' + LastName AS FullName, Name
FROM Departments, Teachers
WHERE Teachers.DepartmentId = Departments.Id;
```

Below the query, under the "Messages" tab, there is an error message:

```
Msg 209, Level 16, State 1, Line 1
Ambiguous column name 'Id'.
```

Рисунок 3.3. Ошибка: двусмысленность при использовании столбца Id

Данная ошибка произошла из-за наличия столбца с одинаковым названием (`Id`) и в таблице `Departments`, и в таблице `Teachers`. Для того чтобы устранить ошибки, связанные с одинаковым названием столбцов в различных таблицах, необходимо указывать полное имя столбца, которое состоит из названий таблицы и столбца, разделенных точкой, например, в нашем случае: `Departments.Id`.

Также вместо полного имени таблицы можно назначить ей псевдоним при помощи оператора `AS`, который рассматривался нами в уроке №3. Применим полученные знания на практике и перепишем ошибочный SQL-запрос:

```
SELECT FirstName + ' ' + LastName AS FullName,
       Name, D.Id AS DeptIdent
  FROM Departments AS D, Teachers AS T
 WHERE T.DepartmentId = D.Id;
```

Результат этого SQL-запроса представлен на рисунке 3.4.

FullName	Name	DeptIdent
Sophia Nelson	Software development	1
Emma Kirk	Mathematics	2
Henry MacAlister	Software development	1
Michael Cooper	Software development	1
Daniel Williams	Cybersecurity	3

Рисунок 3.4. Использование псевдонимов

При помощи следующего SQL-запроса мы хотим определить какие предметы ведет каждый из преподавателей, в этом случае нам необходимо связать между

собой таблицы **Teachers** и **Subjects**, используя таблицу **TeachersSubjects**:

```
SELECT FirstName + ' ' + LastName AS FullName,
       Name AS SubjectName
  FROM Teachers AS T, Subjects AS S,
       TeachersSubjects AS TS
 WHERE T.Id=TS.TeacherId AND S.Id=TS.SubjectId;
```

Результат выполнения данного SQL-запроса представлен на рисунке 3.5.

FullName	SubjectName
Sophia Nelson	C#
Sophia Nelson	JavaScript
Emma Kirk	Discrete Math
Henry MacAlister	SQL Server
Michael Cooper	ADO.NET
Daniel Williams	ITE1
Daniel Williams	WIN10

**Рисунок 3.5. Получение информации «Преподаватели-Предметы»**

При помощи заключительного SQL-запроса мы хотим получить информацию о том какие предметы читаются на той либо иной кафедре, для чего нам придется установить связь между всеми имеющимися таблицами:

```
SELECT D.Name AS DeptName, S.Name
      AS SubjectName
  FROM Departments AS D, Teachers AS T, Subjects
      AS S, TeachersSubjects AS TS
 WHERE D.Id = T.DepartmentId AND T.Id=TS.
       TeacherId AND S.Id=TS.SubjectId;
```

Результат этого SQL-запроса представлен на рисунке 3.6.

DeptName	SubjectName
Software development	C#
Software development	JavaScript
Mathematics	Discrete Math
Software development	SQL Server
Software development	ADO.NET
Cybersecurity	ITE1
Cybersecurity	WIN10

**Рисунок 3.6.** Получение информации  
«Кафедры-Предметы»

## Декартово произведение

При написании многотабличных запросов особое внимание следует уделять установлению взаимосвязи между таблицами, вы уже видели это в предыдущих примерах, например: `Teachers.DepartmentId = Departments.Id` — равенство соответствующего первичного и внешнего ключей. Такое сравнение является критически важным, если его не указывать, то в результате SQL-запроса вы получите всевозможные сочетания записей одной таблицы со всевозможными записями из другой таблицы, полученное множество называется **декартовым произведением** этих таблиц. Продемонстрируем это на примере:

```
SELECT FirstName + ' ' + LastName
    AS FullName, Name
FROM Departments, Teachers;
```

В результате выполнения этого SQL-запроса у нас получится, что все преподаватели одновременно работают на всех кафедрах, однако это не соответствует

действительности. Полученный результат частично отображен на рисунке 3.7.

FullName	Name
Sophia Nelson	Software development
Emma Kirk	Software development
Henry MacAlister	Software development
Michael Cooper	Software development
Daniel Williams	Software development
Sophia Nelson	Mathematics
Emma Kirk	Mathematics
Henry MacAlister	Mathematics

Рисунок 3.7. Декартово произведение таблиц

## 4. Домашнее задание

1. В домашнем задании к уроку №3 вам необходимо было создать однотабличную базу данных, в которой должна содержаться произвольная информация о некой виртуальной больнице. Мы предлагаем вам:

- a. Создать на ее основе многотабличную базу данных, которая должна соответствовать нормальной форме Бойса-Кодда;
- b. Создать диаграмму этой базы данных;
- c. Написать многотабличные SQL-запросы к этой базе данных:
  - вывести информацию обо всех пациентах, находящихся в больнице;
  - показать данные о пациентах, которые лежат в определенном отделении;
  - получить данные о пациентах, которые лежат в больнице больше месяца, отсортировав их по возрастанию даты поступления;
  - вывести информацию о пациентах, которые были выписаны в прошлом месяце;
  - показать информацию о пациентах, которые лежали в больнице с октября по декабрь прошлого года в определенном отделении;
  - вывести данные о пациентах, которых лечит определенный врач с одинаковыми заболеваниями.





## Урок № 4

# Многотабличные базы данных

© Юрий Задерей.

© Компьютерная Академия «Шаг».

[www.itstep.org](http://www.itstep.org)

Все права на охраняемые авторским правом фото-, аудио- и видеопрограммные средства, фрагменты которых использованы в материале, принадлежат их законным владельцам. Фрагменты произведений используются в иллюстративных целях в объеме, оправданном поставленной задачей, в рамках учебного процесса и в учебных целях, в соответствии со ст. 1274 ч. 4 ГК РФ и ст. 21 и 23 Закона Украины «Про авторське право і суміжні права». Объем и способ цитируемых произведений соответствует принятым нормам, не наносит ущерба нормальному использованию объектов авторского права и не ущемляет законные интересы автора и правообладателей. Цитируемые фрагменты произведений на момент использования не могут быть заменены альтернативными, не охраняемыми авторским правом аналогами, и как таковые соответствуют критериям добросовестного использования и честного использования.

Все права защищены. Полное или частичное копирование материалов запрещено. Согласование использования произведений или их фрагментов производится с авторами и правообладателями. Согласованное использование материалов возможно только при указании источника.

Ответственность за несанкционированное копирование и коммерческое использование материалов определяется действующим законодательством Украины.