ЦИФРОВАЯ КУЛЬТУРА
УНИВЕРСИТЕТ ИТМО

Проектирование данных

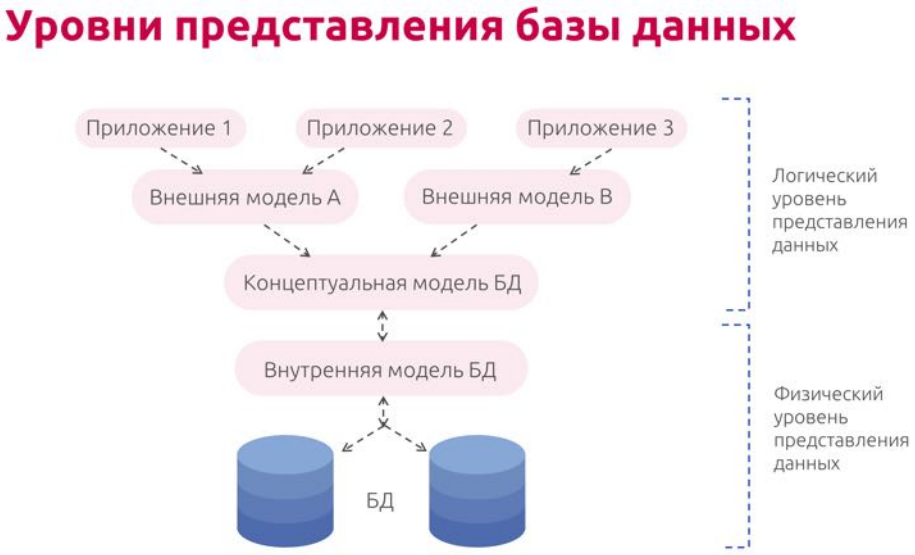
Высшая Школа Цифровой Культуры
Университет ИТМО
dc@itmo.ru

Содержание

1 Диаграммы сущность-связь (ER-диаграммы). Сущности.

Процесс создания базы данных всегда начинается с описания предметной области. Первое, что нужно выяснить – на какие запросы пользователей предстоит ответить информационной системе. Необходимое описание структуры информационных объектов или понятий, которые будут храниться, нужно понять, как они будут связаны между собой, и какие требования предъявляются к допустимым значениям данных.

Проект базы данных надо начинать с анализа предметной области и выявления требований к ней отдельных пользователей (например, сотрудников организации, для которых создается база данных).

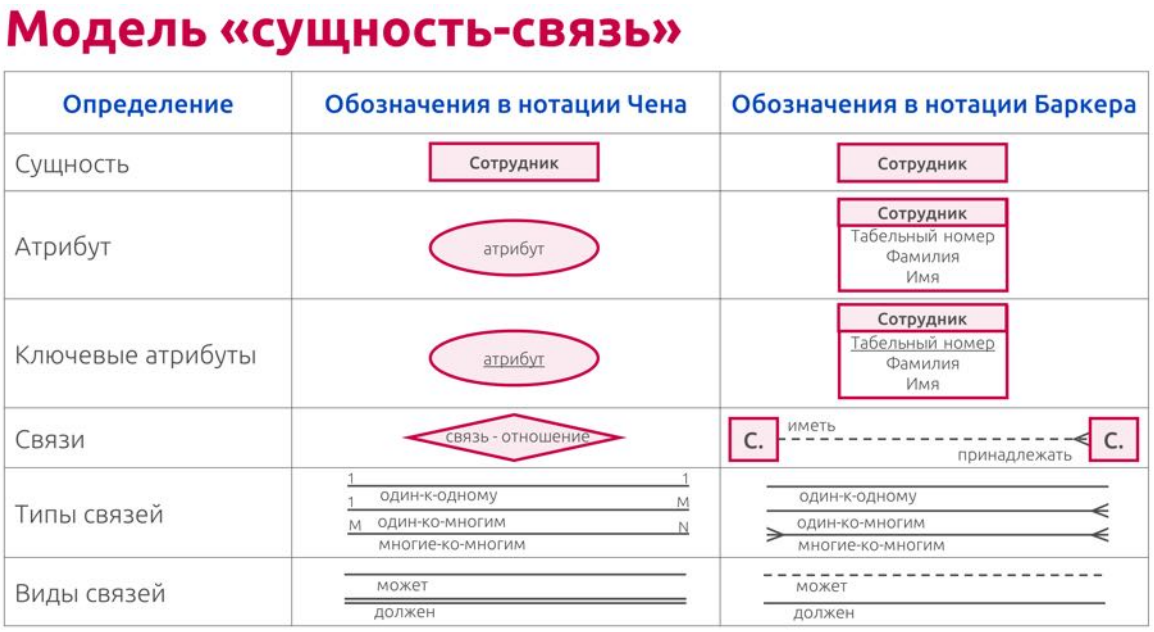


Объединяя частные представления о содержимом базы данных, полученные в результате опроса пользователей, и свои представления о данных, которые могут потребоваться в будущих приложениях, проектировщик сначала создает обобщенное неформальное описание создаваемой базы данных. Это описание, выполненное с использованием естественного языка, математических формул, таблиц, графиков и других средств, понятных всем людям, работающих над проектированием базы данных, называют концептуальным уровнем.

Концептуальный уровень описывает полную логическую структуру хранимых данных. На последнем шаге проектирования создается внутренняя модель данных, которая, которая описывает представление концептуальной схемы в контексте выбранной СУБД.

Для представления данных на концептуальном уровне нужно удобное средство, которое позволяет наглядно отобразить структуры данных предметной области и связи между ними. Подходящим средством для этого явля-

ются диаграммы модели «Сущность-Связь», или «Entity-Relationship». Впервые описание таких диаграмм было предложено Питером Ченом в 1976 г.



В дальнейшем было разработано много подобных моделей, имеющих похожие обозначения. Все варианты диаграмм сущность-связь исходят из одной идеи – рисунок всегда нагляднее текстового описания. Рассмотрим построение диаграмм в нотации Баркера.

Элементы предметной области могут быть представлены как совокупность объектов и связей между ними. Основными элементами диаграмм являются сущности, под которыми понимают хранимые объекты или понятия, атрибуты, которые представляют свойства сущностей, и связи между сущностями.

Давайте для примера рассмотрим базу данных `Экзаменационная сессия`. В базе данных должна храниться информация о студентах, сданных ими экзаменах по определенным предметам. Студент характеризуется номером зачетки, ФИО, датой рождения. Может иметь несколько телефонов. Все студенты распределены по учебным группам. В каждой группе выбран ее староста.

Для группы назначаются экзамены по определенным курсам, экзаменаторы, даты проведения экзаменов и аудитории. За каждый сданный экзамен студенту ставится оценка.

Ограничения: номер зачетки – положительное число, в ФИО не бывает цифр, оценка бывает от двух до пяти.

Сущность

Сущность – это реальный или абстрактный объект определенного вида, который может существовать независимо от других. Иногда сущностью называют класс однотипных объектов, информация о которых будет храниться в базе данных.

У каждой сущности должно быть уникальное имя, к одному и тому же имени всегда должна применяться одна и та же интерпретация. Набор экземпляров сущностей образует множество.

Сущности в ER-диаграммах изображаются с помощью прямоугольников, где внутри наверху указывают название сущности. Название сущности обычно имя существительное.

В базе данных `Экзаменационная сессия` можно выделить сущности `Студент`, `Преподаватель`, `Экзамен`, `Группа`, `Предмет`.

Атрибут

Атрибут – поименованная характеристика сущности. Его наименование должно быть уникальным для конкретного типа сущности, но может быть одинаковым для различного типа сущностей.

Каждый атрибут имеет свой тип данных. Например, у сущности `Студент` атрибутами могут быть имя, фамилия, серия и номер паспорта, номер зачетки, дата рождения. `Группа` будет описана номером и специализацией. `Преподаватель` – ФИО и должность.

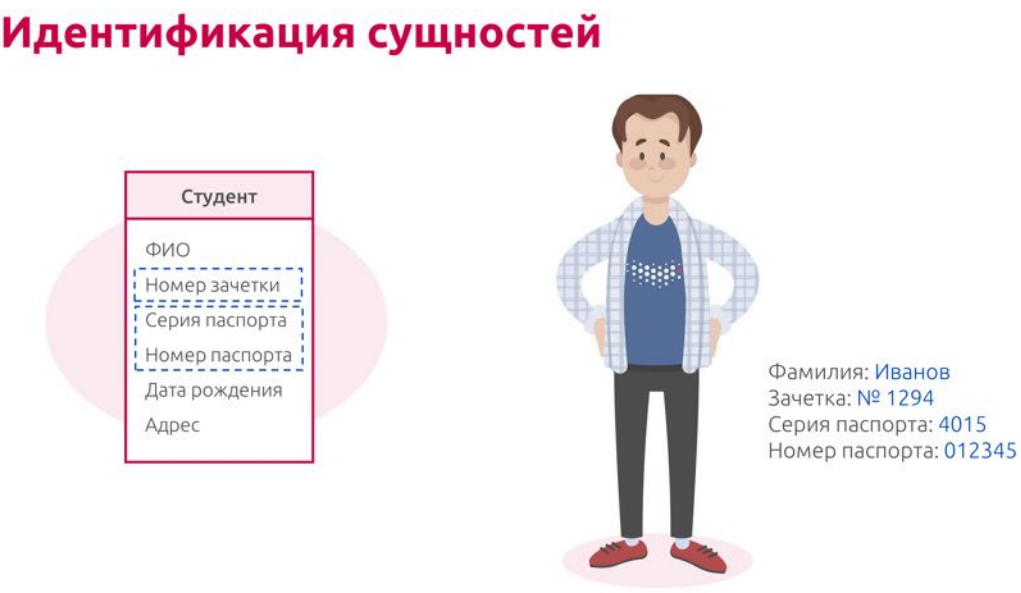
Идентификация сущностей

Экземпляры сущностей должны быть отличны друг от друга, а значит нужны **идентификаторы сущностей**. Экземпляры сущностей не могут быть дубликатами, поэтому как минимум, значения всех атрибутов образуют уникальную комбинацию.

Но чаще всего бывает так, что не все атрибуты нужны для идентификации сущности. Находятся один или несколько атрибутов, необходимых и достаточных для этого. Например, чтобы точно указать студента, нужно знать его номер зачетки, а дата рождения и адрес для этого не нужны.

На ER-диаграммах отображаются и ключи. Напомним, что ключом называют один или несколько атрибутов сущности, по которым можно однозначно идентифицировать сущность.

Если ключом является один атрибут, то он называется простым. Ключ, составленный из нескольких атрибутов, называют составным. Иногда у сущности может быть несколько возможных ключей.



Примерами для сущности студент может быть простой ключ номер зачетки, или составной ключ – комбинация атрибутов серия и номер паспорта.

При наличии нескольких возможных ключей один из них, наиболее часто используемый при поиске, называют первичным ключом. Хорошим тоном является выбор в качестве первичного ключа числового, короткого и не

меняющегося в процессе жизни объекта атрибута. На диаграмме атрибуты, входящие в состав первичного ключа, подчеркивают одинарной линией.

Есть несколько возможных способов идентификации:

- естественные ключи;
- «по положению» (географическое, по порядку, по времени);
- суррогатные.

Обычно сущность обладает атрибутами, характеризующими ее свойства, которые идентифицируют каждый экземпляр сущности. Такие ключи называют естественными.

Ни один естественный идентификатор не может быть абсолютно надежен, поэтому для системы удобнее суррогатные ключи. Но они бесполезны при поиске.

Не всегда среди атрибутов сущности легко подобрать такой атрибут, а иногда его просто нет. Например, у сущности `Предмет` – название в качестве ключа нельзя взять, т.к. могут быть предметы с одинаковым названием, вести предмет могут разные преподаватели.

Если нет естественного ключа, придумывают искусственный – «суррогатный». Для сущности Предмет можно придумать такой числовой идентификатор.

При построении ER-диаграммы иногда трудно принять решение, где атрибут сущности, а где другая отдельная сущность.

Абсолютное различие между типами сущностей и атрибутами отсутствует. Атрибут является таковым только в связи с типом сущности. В другом контексте атрибут может выступать как самостоятельная сущность. Например, для человека адрес может быть атрибутом, если нас интересует только место прописки.

Но адрес может быть и самостоятельным объектом, имеющим свои характеристики Город, улица и номер дома. Тогда мы сможем узнать не только

где человек живет, но и отвечать на более сложные запросы: сколько человек живет в таком-то городе или на определенной улице. Поэтому выбор между возможным атрибутом или новой сущности нужно делать исходя из возможных запросов к данным.

2 Диаграммы сущность-связь (ER-диаграммы). Связи.

Мы разобрались с проектированием сущностей, теперь рассмотрим отношения, которые возникают между сущностями – их называют связями.

Связь – ассоциирование двух или более сущностей. Если бы назначением базы данных было только хранение отдельных, не связанных между собой данных, то ее структура могла бы быть очень простой. А так как в реальных базах данных нередко содержатся сотни или даже тысячи сущностей, то теоретически между ними может быть установлено более миллиона связей.

Наличие такого множества связей и определяет сложность инфологических моделей. У связей, как и у сущностей, есть свое название – чаще всего глагол. Например, `Студент` *состоит в* `Группе`, `Преподаватель` *принимает* `Экзамен`.

Основное отличие связей от сущностей: связи сами по себе не имеют

смысла и не могут существовать без связываемых сущностей.

Связи, как и сущности, нуждаются в идентификации. Идентификатор связи включает ключи связываемых сущностей и, возможно, некоторые выделенные атрибуты связи

Приведем пример атрибута связи. Например, если студент сдал преподавателю экзамен по некоторому предмету, то оценка, полученная за экзамен, будет атрибутом связи, а не студента, предмета и тем более преподавателя.

Свойства связей

Связи могут отличаться своими характеристиками, наиболее важными из которых являются:

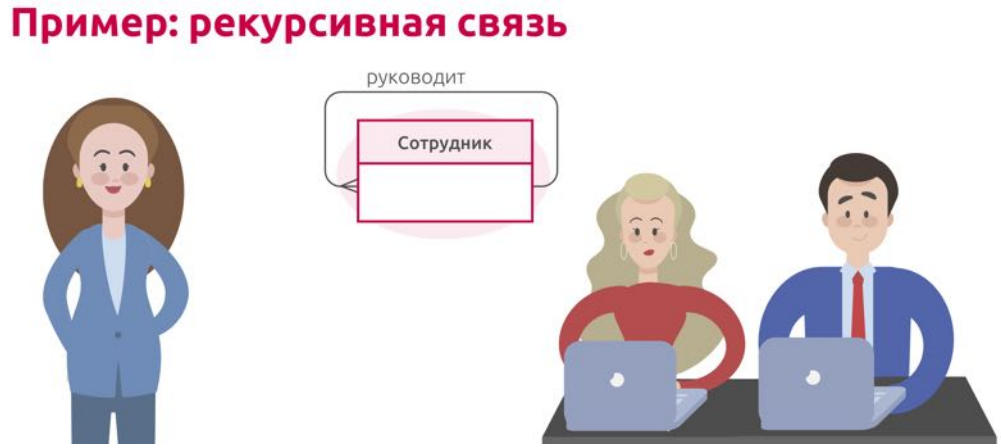
- размерность (степень связи);
- мощность (кардинальность);
- модальность

Степень связи — это количество сущностей, между которыми возникает связь. Самая типовая связь – бинарная, когда связь определена между двумя сущностями.

Если три сущности образовали связь, как в примере с экзаменом: студент, преподаватель и предмет, то связь называется тернарной. Бывают унарные, или рекурсивные связи, когда связь возникает внутри одной сущности – например, один сотрудник руководит другими. В общем случае связь между n сущностями называется n-арной.

Рассмотрим **бинарную связь** между студентом и группой. Между двумя сущностями может быть определено несколько наборов связей. Например, студент может *состоять в* группе, а может *является старостой* группы, это разные связи между одними и теми же сущностями.

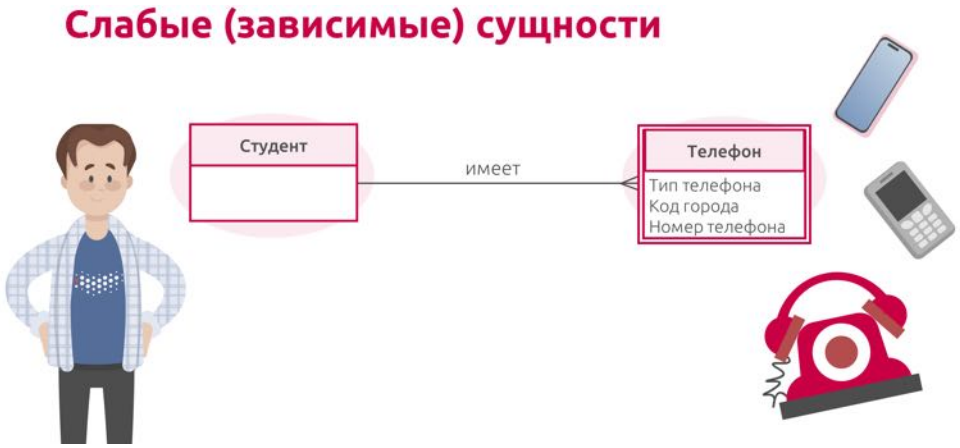
Примером **тернарной связи** может быть экзамен: в этой связи участвуют три сущности – `Студент`, `Преподаватель` и `Предмет`. Оценка является собственным атрибутом связи. Часто для удобства вместо n-арных связей используют только бинарные. Для этого нам придется вместо связи *экзамен* ввести сущность `Экзамен`.



Приведем пример **рекурсивной связи**. Например, рассмотрим сущность `Сотрудник`. Одни сотрудники могут руководить другими, в то же время оставаясь сотрудниками. Получается, что сущность вступает в связь с другими экземплярами той же сущности.

Бывают «правильные» (независимые) и «слабые» сущности. Слабые сущности не могут существовать без связи с другими (сильными) сущностями, или находятся в зависимости от других сущностей.

Давайте представим, что нам нужно хранить информацию о телефонах студента. На первый взгляд, может показаться, что телефонный номер — это просто атрибут. Но у студента может быть несколько телефонных номеров – домашний, мобильный, а может быть, и служебный, и мобильных может быть несколько. Тогда правильнее выделить телефон в отдельную сущность. У сущности `Телефон`, кроме номера, могут быть еще атрибуты: тип телефона, код города и пр. У нескольких людей телефонные номера могут совпасть, – например, домашний и рабочий. Формально чтобы отличать их, нужно



придумать какой-то идентификатор, или суррогатный ключ. Но навряд ли нам будет нужен телефонный номер сам по себе, без информации о том, кто его владелец.

В таком случае информация о слабой сущности хранится вместе со связью, которая указывает на независимую сущность, и суррогатный ключ не нужен. Такие сущности и связи на диаграммах иногда очерчивают двойной линией.

Мощность обозначает максимальное количество экземпляров одной сущности, связанных с экземплярами другой сущности.

Делятся на три вида в зависимости от количества участвующих в них экземпляров сущностей.

- один-к-одному `1:1`
- один-ко-многим `1:N`
- многие-ко-многим `M:N`

Мощность обозначает максимальное количество экземпляров одной сущности, связанных с экземплярами другой сущности. Мощность связи на диа-

граммах отражает конец линии, соединяющий сущности, который называют «концом связи». Связь один обозначает прямая линия на конце связи, связь много – перевернутая стрелка, которую часто называю вороньей лапкой Crow's Foot.

Иногда в СУБД на диаграммах отображают связь «один» в виде ключа, а связь «много» в виде знака бесконечность.

Связи один-к-одному

Наиболее просто описываемая, но реже всего встречаемая в природе, это связь вида один к одному. Она возникает, когда ровно один экземпляр одной сущности вступает в связь с одним и только одним экземпляром другой сущности.

В качестве примеров можно привести связь между столичным городом и страной, где город является столицей, ректор – вуз для связи Руководит. Ромб связи и прямоугольник объекта соединяются прямой линией.

Связи один-ко-многим

Один-ко-многим: каждому экземпляру первой сущности может соответствовать несколько экземпляров другой сущности, но каждому экземпляру второй сущности соответствует не более одного экземпляра первой сущности.

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

Связи многие-ко-многим

Встречаются также связи вида многие-ко-многим: каждому экземпляру первой сущности может соответствовать несколько экземпляров другой сущности, и наоборот.

Модальность связей

Последнее свойство связей, которое мы рассмотрим, называется **модальность**. Необязательные связи (условные) – модальность «может» означает, что экземпляр одной сущности может быть связан с одним или несколькими экземплярами другой сущности, а может быть и не связан ни с одним экземпляром.

- Пассажир может иметь билет.
- Человек может иметь автомобиль.
- Студент может сдать экзамен.

Обязательные – модальность «должен» означает, что экземпляр одной сущности обязан быть связан не менее чем с одним экземпляром другой сущности.

- Каждый курс лекций должен иметь преподавателя.
- У каждой кафедры должен быть заведующий.
- Каждый билет выписан на имя конкретного пассажира.

Если рассмотреть связь между студентом и экзаменом, то тут должна быть модальность «может», т.к. студент может не сдать ни одного экзамена, и может найтись экзамен, который никто еще не сдал.

Например: каждый сотрудник может участвовать в нескольких проектах, и в каждом проекте участвуют несколько сотрудников. Связь студент-курс: каждый студент изучает множество курсов, и каждый курс изучается многими студентами.

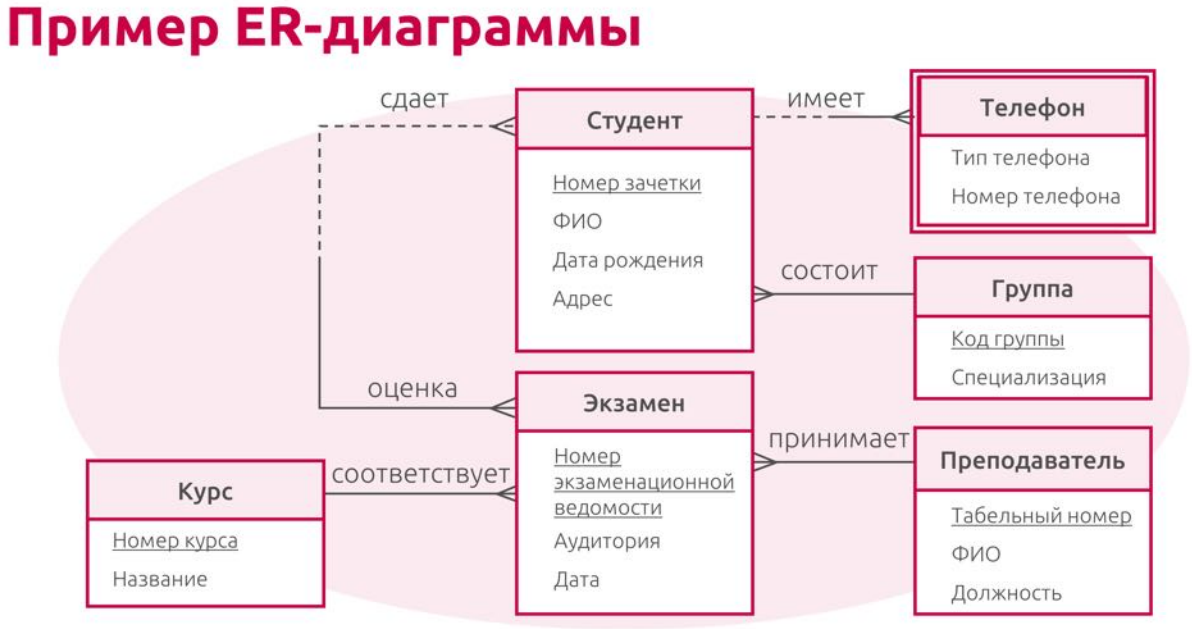
Таким образом мы рассмотрели все элементы ER-диаграмм.

Итак, чтобы построить диаграмму, нужно сделать следующие шаги:

1. Определить сущности.
2. Определить атрибуты сущностей.
3. Определить первичные ключи.
4. Определить отношения между сущностями.
5. Определить кардинальность.
6. Нарисовать ER-диаграмму.
7. Проверить ER-диаграмму.

Вспомним предметную область, которую мы рассматривали: `Экзаменационная сессия`. В нашей базе данных должна храниться информация о студентах, сданных им экзаменах по определенным предметам. Студент характеризуется номером зачетки, ФИО, датой рождения. Может иметь несколько телефонов. Все студенты распределены по учебным группам. Для группы назначается экзамен по определенному предмету, назначается экзаменатор. Указывается дата экзамена и аудитория. За каждый сданный экзамен студенту ставится оценка. Ограничения: номер

зачетки – положительное число, в ФИО не бывает цифр, оценка бывает от двух до пяти. Студент состоит в одной учебной группе, в один день может сдать только один экзамен.



3 Преобразование ER-модели в реляционную базу данных

Мы научились определять структуры данных в терминах модели сущность-связь при помощи ER-диаграмм. Следующий шаг после этого выбор конкретной СУБД, поддерживающей некоторую модель данных, и отображение диаграмм на структуры, принятые в выбранной СУБД. Если выбором будет реляционная СУБД, то необходимо преобразовать структуры данных ER-модели в отношения.

Давайте вспомним, что отношения — это объекты базы данных, в которых хранится вся доступная пользователю информация.

Отношение имеет заголовок, или схему с описанием структуры таблицы, состоящий из атрибутов, или названий столбцов, и тело отношения, составленное из строк, или кортежей – в них содержатся непосредственно данные. Значения каждого атрибута содержатся в допустимом множестве возможных значений, которое называется доменом.

Схема отношения – упорядоченное множество атрибутов. Каждый атрибут имеет имя, уникальное в рамках отношения, и для каждого из которых задан тип данных.

Типы данных

В разных СУБД могут быть некоторые отличия в поддерживаемых типах данных и в их наименовании, но в любой СУБД можно хранить числа, строки и информацию о дате и времени.

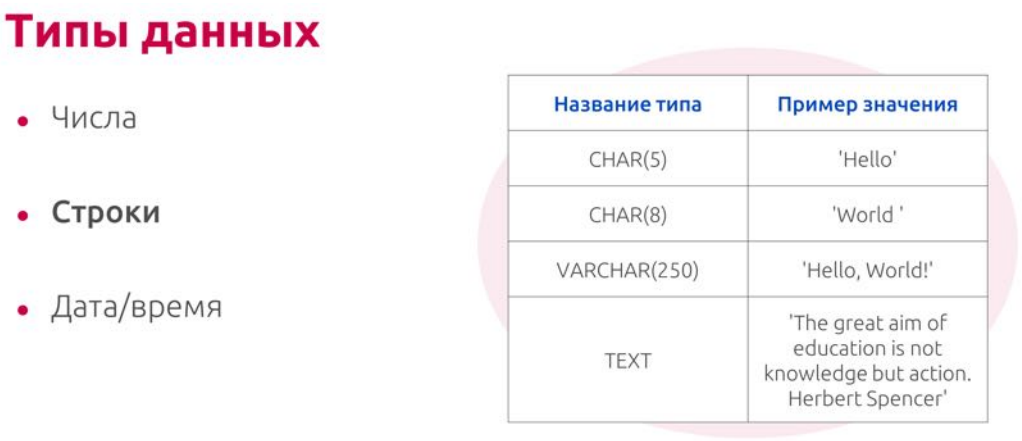
Типы данных

- Числа
- Строки
- Дата/время

Название типа	Пример значения
NUMBER	23
NUMERIC(4,2)	67.13
INTEGER	1000000090
DECIMAL (20,10)	0.0000000003
FLOAT	4.248095
REAL	- 3.49E+10
FLOAT	- 8.23E-40

Числовые типы данных могут быть целыми и дробными, заданы с разной точностью, т.е. можно явно указывать количество знаков в целой и дробной частях числа.

Символьные данные могут быть представлены строками фиксированной или переменной длины. При описании типа в скобках указывают размер. Для строк переменной длины этот размер задает максимально возможную длину строки, а для строк фиксированной длины – точную длину строки. Если



не указать размер, то в некоторых СУБД длина поля будет равна одному символу, в некоторых – строке переменной длины максимально возможного размера.

Тип данных дата и время также используется в любой СУБД, но именно для этого типа чаще всего проявляются особенности реализации. Для даты и времени есть много разных форматов, например, хранение в одном типе даты и времени или по отдельности.

В ряде СУБД предусмотрена специальные типы данных для хранения данных в других форматах: пространственных и иерархических данных, данных в формате XML и JSON и другие.

Для каждой сущности, описанной в ER-модели, в базе данных нужно создать отдельную таблицу. Имя сущности становится именем таблицы. Каждый атрибут становится столбцом. Для имен таблиц и столбцов принято использовать латинские буквы. Для каждого столбца определяется подходящий тип данных. Идентификатор сущности превращаются в ключ таблицы.

Напомним, что возможным ключом называют минимальный набор атрибутов, по которому можно определить все остальные. В качестве первичного ключа выбирают один из возможных ключей, обычно чаще всего используемый при поиске.

Например, в базе данных `Экзаменационная сессия` нужно создать таблицу для сущности `Студент` с атрибутами номер зачетки, ФИО, серия и номер

паспорта, дата рождения и адрес. Создадим таблицу `STUDENT` с полями:

- Номер зачетки – `StudentId` – целое число;
- ФИО – `StudentName` – строка переменной длины до 100 символов;
- Серия паспорта – `Pass_s` – строка до 4 символов;
- Номер паспорта – `Pass_num` – строка до 6 символов;
- Дата рождения – `BirthDate` – формат Дата;
- Адрес – `Address` – строка переменной длины до 100 символов.

Возможные ключи – номер зачетки и серия и номер паспорта. Выберем в качестве ключа поле `StudentId`.

Создадим таблицу для сущности `Группа` назовем `ST_GROUP`. Полями будут:

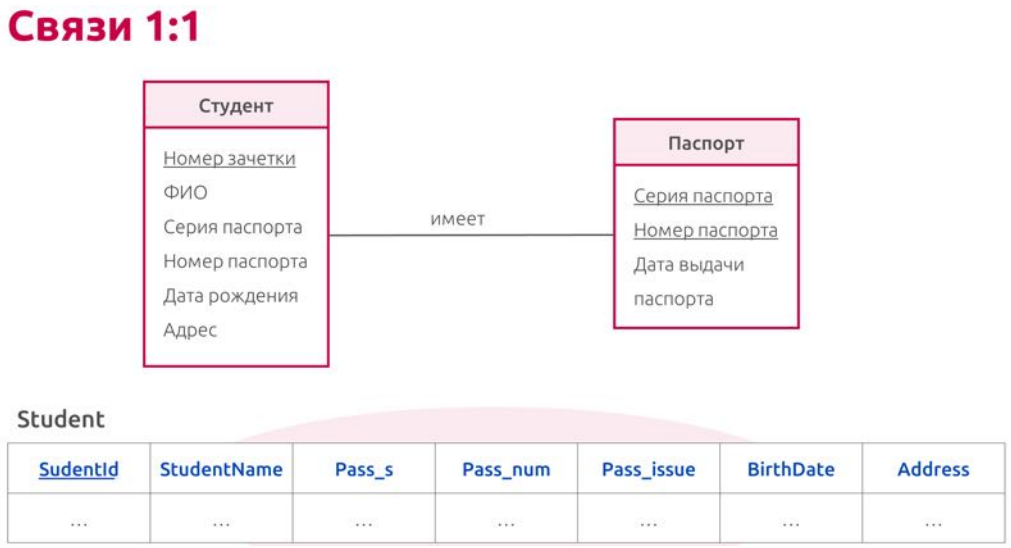
- Код группы – `GroupCode` – строка до 32 символов;
- Специализация – `Specialization` – строка переменной длины до 100 символов.

Ключ – `GroupCode`.

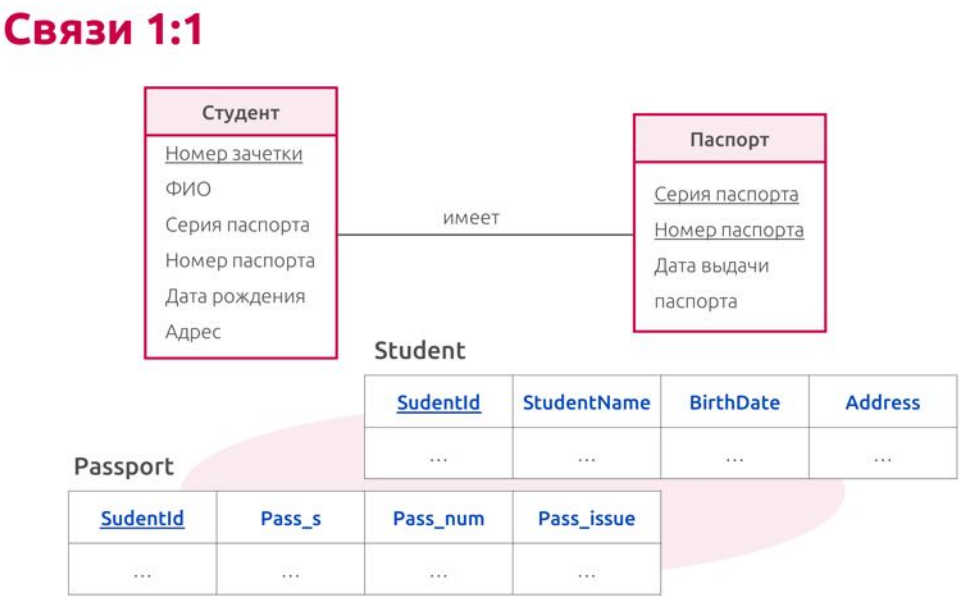
Связи также хранятся в отношениях. Для отображения связи необходимы ключевые атрибуты объектов, участвующих связи, и атрибуты связи, если таковые есть. Связи вида один к одному принято хранить в одной таблице, столбцы которой соответствуют атрибутам обеих сущностей.

Рассмотрим простейший пример связи вида `1:1`. Предположим, что требуется хранить информацию о паспортных данных студентов. Тогда мы бы ввели отдельную сущность – номер паспорта с атрибутами серия, номер и дата выдачи.

У каждого студента ровно один паспорт, и каждый паспорт соответствует ровно одному человеку. В таком случае можно было эту информацию можно представить в одной таблице `Студент` с атрибутами Идентификатор студента, ФИО, Серия паспорта, Номер паспорт, Дата выдачи паспорта, Дата



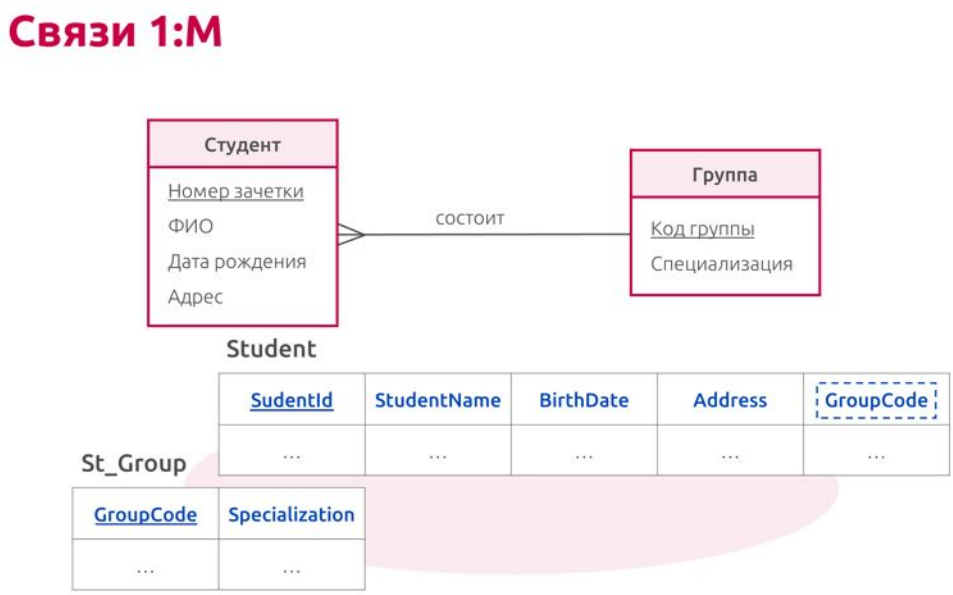
рождения, Адрес. Ключом отношения может быть Идентификатор студента или серия и номер паспорта.



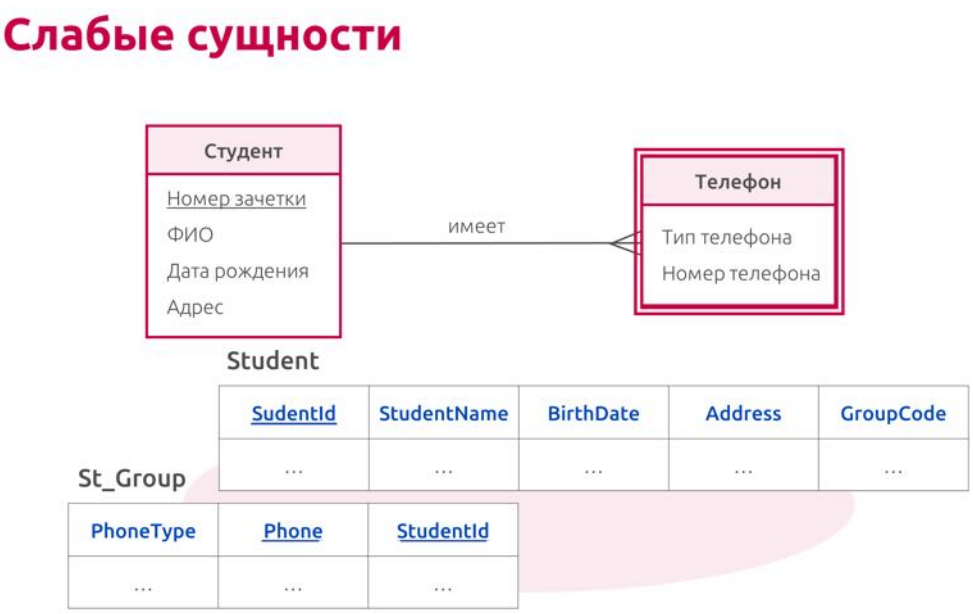
Иногда, наоборот, атрибуты одной сущности хранят в двух разных таблицах. Например, из соображений безопасности мы могли бы хранить информацию о номерах паспортов студентов в отдельной таблице.

Тогда в одной таблице были бы поля Номер зачетки – `StudentId`, `StudentName` – имя студента, дата рождения и адрес. Ключ – `StudentID`. В другой таблице хранились бы `Pass_s` серия паспорта и `Pass_num` номер паспорта и `StudentId` – ключ.

Если между сущностями связь вида один ко многим, как например, между сущностями `Группа` и `Студент` – в каждой группе учится один или более (много) студентов, и каждый студент учится ровно в одной группе, то реализовать эту связь можно следующим образом: в таблицу сущности с концом



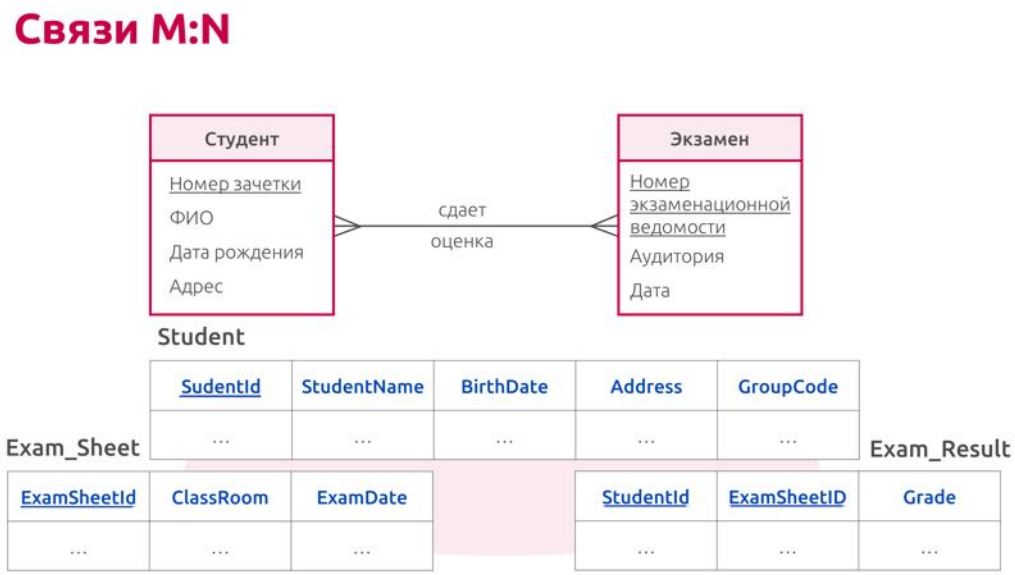
связи «много» добавляем ключ сущности с концом связи «один».



Также хранится информация о связях с слабыми сущностями. У студента может быть несколько телефонных номеров, и у некоторых номера могут совпасть, например, домашние.

Мы добавим к таблице с информацией о номерах телефона ключ сущности `Студент` – `StudentID`. Ключ отношения будет составным: `Phone` и `StudentID`.

Для хранения связей вида многие ко многим приходится создавать отдельную таблицу, которая служит для отображения связи. Схема данного отношения составляется из ключевых атрибутов объектов, участвующих в связи, и атрибутов связи, если таковые есть. Например, связь между сущностью `Студент` и `Экзамен` имеет вид многие-ко-многим. За экзамен студент



получает оценку – атрибут оценка является свойством связи. Создадим таблицу `Результат экзамена` с полями Идентификатор студента, Идентификатор экзамена и оценка – целые числа.

4 Создание таблиц

Мы рассмотрели, как можно отобразить концептуальную модель базы данных в виде ER-диаграммы на реляционную, табличную модель. На следующем этапе нужно описать таблицы и ограничения целостности в контексте конкретной СУБД. Для этого в любой СУБД существует специальный язык, который дает возможность создавать объекты и работать с ними. Наиболее распространенный – структурированный язык запросов `SQL`. Именно

он чаще всего используется для работы с реляционными базами данных и поддерживается большинством производителей СУБД. Поэтому мы будем рассматривать именно его.

Все операторы языка можно поделить на три группы:

- операторы определения данных;
- операторы манипулирования данными;
- операторы управления данными.

Для описания структуры таблиц и других объектов нужно использовать операторы DDL – язык определения данных. Этих операторов всего три: операторы `CREATE`, `ALTER` и `DROP` – для создания, изменения и удаления объектов базы данных. Опции операторов существенным образом зависят от специфи-

SQL DDL (Язык Определения Данных)

- *CREATE* <OBJECT> [OPTIONS] - создание
- *ALTER* <OBJECT> [OPTIONS] - изменение
- *DROP* <OBJECT> [OPTIONS] - удаление

ки создаваемого объекта, и от контекста используемой СУБД.

Язык манипулирования данными содержит операторы, позволяющие отображать и модифицировать содержимое таблиц: добавлять и изменять строки, удалять и искать их по заданным критериям.

DML (Язык Манипулирования Данными)

- *SELECT* - поиск
- *INSERT* - вставка
- *UPDATE* - изменение
- *DELETE* - удаление
- *TRUNCATE* - удаление

Язык контроля данных состоит из операторов, которые управляют правами пользователя в базе данных. Например, разрешить ли пользователю прочитать данные из таблицы, исполнить хранимую процедуру и т.п.

DCL (Язык Управления Данными)

- *GRANT* - назначение привилегии
- *REVOKE* - отзыв привилегии

СУБД предоставляют среду для работы с базой данных, в которой можно писать и выполнять команды. Создание базы данных начинается со специ-

Создание базы данных

- *CREATE DATABASE db_name [options];*
- *CREATE USER [options];*
- *CREATE SCHEMA [options];*

альной команды, при выполнении которой резервируется некоторая область хранилища, где будут сохраняться все объекты базы. Конечно, при создании можно указать не только имя базы данных, а место ее хранения, задать форматы по умолчанию и указать еще много полезных параметров.

Теперь можно приступить к созданию таблиц. Нам понадобится оператор `CREATE TABLE`, и затем в круглых скобках нужно описать создаваемую таблицу. Описание таблицы состоит из описания столбцов и ограничений целостности. Столбцы таблицы служат для представления атрибутов хранимых сущностей. Ограничения целостности — это правила, которым должны удовлетворять данные. Например, в фамилии не бывает цифр, оценка за экзамен может быть от 2 до 5. Очень важно при описании таблицы описать, какие значения могут принимать хранимые данные, чтобы предотвратить попадание в базу некорректных значений. Чтобы создать простую таблицу, нужно в

Создание таблиц

- *CREATE TABLE <имя таблицы>{*
  {
  {имя столбца}
  {тип данных}
  [значение по умолчанию]
  [список правил целостности]
  }+
)

круглых скобках просто описать столбцы, указывая название столбца и тип.

К примеру, опишем столбцы таблицы `STUDENTS`. В результате выполнения оператора `CREATE TABLE` будет создана пустая таблица с пятью столбцами.

```
CREATE TABLE STUDENTS (
        StudentId INTEGER,
        StudentName VARCHAR(100),
        GroupCode VARCHAR(32),
        BirthDate DATE,
        Address VARCHAR(100)
);
```

StudentId	StudentName	GroupCode	BirthDate	Address

После того, как таблица создана, её структуру можно изменить. Для этого используют оператор `ALTER TABLE`. После `ALTER TABLE` указывается название таблицы, которую требуется изменить, а дальше описывается действие, которое нужно предпринять для изменения таблицы. Чтобы добавить колон-

ку, используется фраза `ADD COLUMN`, в которую передается название колонки.

К примеру, можно добавить номер телефона – поле `Phone_number`, тип данных `CHAR(11)`, затем сделать длину поля 10 символов, и удалить столбец.

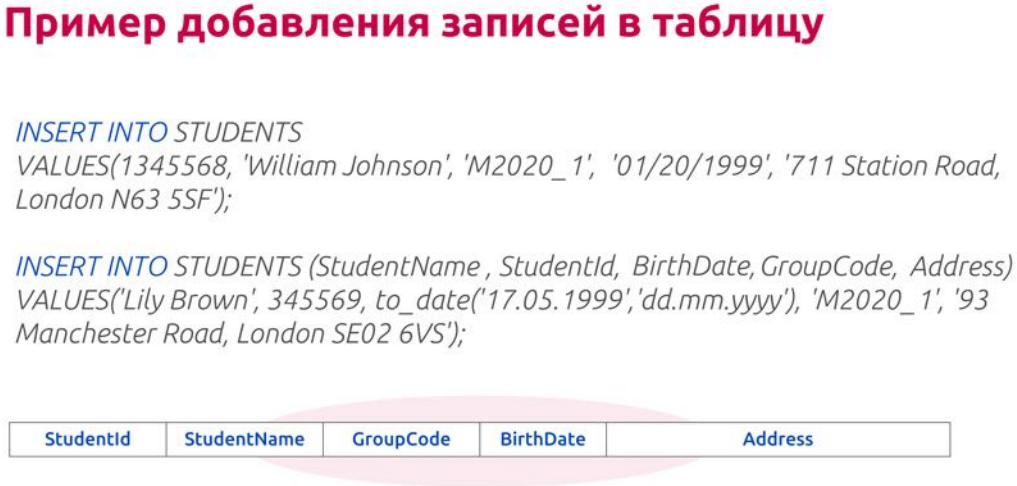
```
ALTER TABLE STUDENTS
ADD COLUMN Phone_number CHAR(11)

ALTER TABLE STUDENTS
ALTER COLUMN Phone_number char(10);

ALTER TABLE STUDENTS
DROP COLUMN Phone_number
```

Удалить таблицу можно с помощью оператора `DROP TABLE`. Для добавления записей используется оператор `INSERT INTO`.

Следующий пример демонстрирует добавление записей в таблицу `STUDENTS`. В первый раз названия столбцов не были указаны, т.к. добавляемые данные соответствовали порядку столбцов, заданному при создании таблицы. Во второй раз значения полей `StudentId` и `StudentName` были указаны не в том порядке, поэтому после названия таблицы был указан порядок полей добавляемых значений. У столбца `BirthDate` тип данных – дата. При вводе



значений этого типа, надо знать установленный в системе формат по умолчанию – даты могут быть представлены в разном формате. Иногда бывает трудно определить, в каком именно. В этой ситуации на помощь может прийти функция преобразования строки в дату. В некоторых СУБД это функция `to_date`. В качестве параметров функции указывают собственно дату в виде строки и ее формат.

Есть два оператора, которые можно использовать для удаления записей: `DELETE` и `TRUNCATE`. Первый оператор удаляет из таблицы строки, соответствующие указанным критериям, а второй удаляет все записи из таблицы. В качестве примера приведем команду для удаления из таблицы `STUDENTS`

Пример удаления записей из таблицы

```
DELETE FROM STUDENTS WHERE StudentId = 1294;

TRUNCATE TABLE STUDENTS ;
```

StudentId	StudentName	GroupCode	BirthDate	Address
...	...	...	...	...
1294	Thomas Wood	B2020_2	12/09/1999	951 Highfield Road, London E16 2RI
...	...	...	...	...

строки с параметром `StudentId = 1294`. Если выполнить второй оператор, указанный на рисунке, то из таблицы `STUDENTS` будут удалены все записи.

Для редактирования записей используется оператор `UPDATE`.

Изменение записей в таблице

```
UPDATE <имя таблицы>
    SET column_1 = expression_1,
        ...
        column_m = expression_m
[WHERE <условие>]
```

Например, изменим адрес у студента с полем `StudentId = 345569`. Обратите внимание, что при помощи оператора `UPDATE` можно изменить значение не только одного поля, а сразу нескольких полей, например адрес и имя, как показано во второй команде.

Пример изменения записей в таблице

```
UPDATE STUDENTS SET Address = '93 Manchester Road, London SE02 6VS'
WHERE StudentId = 345569;

UPDATE STUDENTS SET Address = '93 Manchester Road, London SE02 6VS',
StudentName='Lily Smith'
WHERE StudentId = 345569;
```

StudentId	StudentName	GroupCode	BirthDate	Address
345569	Lily Smith	M2020_1	05/17/1999	93 Manchester Road, London SE02 6VS

5 Ограничения целостности

Мы научились создавать таблицы и изменять их структуру. Для каждого столбца таблицы указывается название столбца и тип данных.

Также мы научились заполнять таблицы значениями при помощи оператора `INSERT`. При добавлении записей в таблицу производится автоматическая проверка соответствия добавляемых таким образом мы можем быть уверены, что в числовое поле не попадет буква, и в поле с типом дата нельзя ввести 32 число 15 месяца. Но может потребоваться задать более серьезные ограничения на возможные значения данных, чем просто соответствие типу данных.

Пример создания таблицы

```
CREATE TABLE STUDENTS (
    StudentId INTEGER,
    StudentName VARCHAR(100),
    GroupCode VARCHAR(32),
    BirthDate DATE,
    Address VARCHAR(100)
);
```

StudentId	StudentName	GroupCode	BirthDate	Address
A1345568	William Johnson	M2020_1	01/20/1999	711 Station Road, London N63 5SF
345569	Lily Brown	M2020_1	32/17/1999	93 Manchester Road, London SE02 6VS
-550	Joy9n Sirl	B2020_2	02/02/1900	951 Highfield Road, London E16 2RI

К примеру, может потребоваться указать, что идентификатор – положительное число, а в фамилии не бывает цифр, некоторые поля могут принимать значения из ограниченного множества – например, пол бывает мужской и женский, дней недели всего 7 и т.д.

Ограничения целостности

```
CREATE TABLE T1 (
    { имя столбца_1 } { тип данных_1} [ список ограничений столбца_1],
    { имя столбца_2 } { тип данных_2} [ список ограничений столбца_2],
    ...
    { имя столбца_n } { тип данных_n} [ список ограничений столбца_n],
    [ список ограничений]
)
```

Для этого помогут ограничения целостности, определяющие возможные значения данных. Существует несколько категорий правил целостности:

- ограничение неопределенных значений;
- ограничение допустимого диапазона значений;

- ограничение уникальности;
- ограничение первичного ключа;
- ссылочная целостность.

Некоторые атрибуты сущности могут иметь неопределенное значение, а другие всегда должны быть заданы. Это обязательные атрибуты, без которых строка таблицы не имеет смысла. Например, сложно представить, что может быть студент, у которого не определено имя или номер зачетки. Для того, чтобы значения поля всегда было определено, для него можно указать ограничение `NOT NULL`.

Ограничение неопределенных значений

```
CREATE TABLE STUDENTS (
    StudentId INTEGER,
    StudentName VARCHAR(100) NOT NULL,
    GroupCode VARCHAR(32),
    BirthDate DATE,
    Address VARCHAR(100)
);
```

Если ограничение `NOT NULL` не задано, то по умолчанию значения полей могут быть не заданы. Это можно прописать в явном виде, например для поля адрес – слово `NULL` рядом с полем означает, что его значения могут быть не определены.

Ограничение повторяющихся значений

```
CREATE TABLE STUDENTS (
    StudentId INTEGER UNIQUE ,
    StudentName VARCHAR(100) NOT NULL,
    GroupCode VARCHAR(32),
    BirthDate DATE,
    Address VARCHAR(100) NULL
    UNIQUE (GroupCode, StudentName, BirthDate)
);
```

Согласно бизнес-логике некоторых данных, значения некоторых атрибутов таблицы должны быть уникальными. Например, уникальным должно быть значение поля `StudentId`, которое соответствует номеру зачетки. Чтобы сделать невозможным повторение значений в столбце, нужно задать ограничение `UNIQUE`.

Иногда уникальным нужно сделать не отдельное поле, а комбинацию полей. Например, потребуем, чтобы в каждой группе не было двух людей с одинаковыми именами и с одной датой рождения.

Когда ограничения затрагивают не одно поле, а несколько, то описание ограничений делают после описания всех полей таблицы. Нужно указать ограничение `UNIQUE`, затем в скобках перечислить поля, значения которых должны быть уникальной комбинацией. Ограничение уникальности поля можно усилить, добавив ограничение `NOT NULL` для поля `StudentId`.

Ограничение первичный ключ

```
CREATE TABLE STUDENTS (
    StudentId INTEGER PRIMARY KEY,
    StudentName VARCHAR(100) NOT NULL,
    GroupCode VARCHAR(32),
    BirthDate DATE,
    Address VARCHAR(100) NULL
    UNIQUE (GroupCode, StudentName, BirthDate)
);
```

Более сильным, чем уникальность, является ограничение первичный ключ. Это ограничение требует уникальности значений поля, отсутствия не заданных значений. Кроме того, первичный ключ в таблице может быть только один. Но следует отметить, что в качестве первичного ключа может быть комбинация нескольких столбцов таблицы.

Ограничение первичный ключ

```
CREATE TABLE EXAM_RESULT
    (StudentId INTEGER NOT NULL,
    ExamSheetId INTEGER NOT NULL,
    Mark INTEGER,
    PRIMARY KEY(StudentId, ExamSheetId)
);
```

Очень важным ограничением, позволяющим хранить связи в таблицах, является ссылочная целостность. Чтобы продемонстрировать ее, рассмотрим еще одну таблицу `ST_GROUP`. Каждый студент относится к определенной группе.

Посмотрим на содержимое таблиц `STUDENTS` и `ST_GROUP`. Вильям и Лили

Ссылочная целостность

Students

StudentId	StudentName	GroupCode	BirthDate	Address
1345568	William Johnson	M2020_1	01/20/1999	711 Station Road, London N63 5SF
345569	Lily Brown	M2020_1	05/17/1999	93 Manchester Road, London SE02 6VS
130568	John Sirl	M2020_2	02/02/1999	951 Highfield Road, London E16 2RI

St_group

M2020_1	Nanotechnology
M2020_1	Health Research
B2020_2	Art&Science

учатся в группе M2020_1. Из таблицы `ST_GROUP` мы можем найти их специализацию. Но как найти специализацию Джона? Он учится в группе M2020_2, но такой группы нет в таблице `ST_GROUP`. Мы видим, что логика данных нарушена. Нельзя указывать для студента группу, которой нет в списке групп. Чтобы такого не происходило, надо связать столбцы двух таблиц соответствующим правилом целостности.

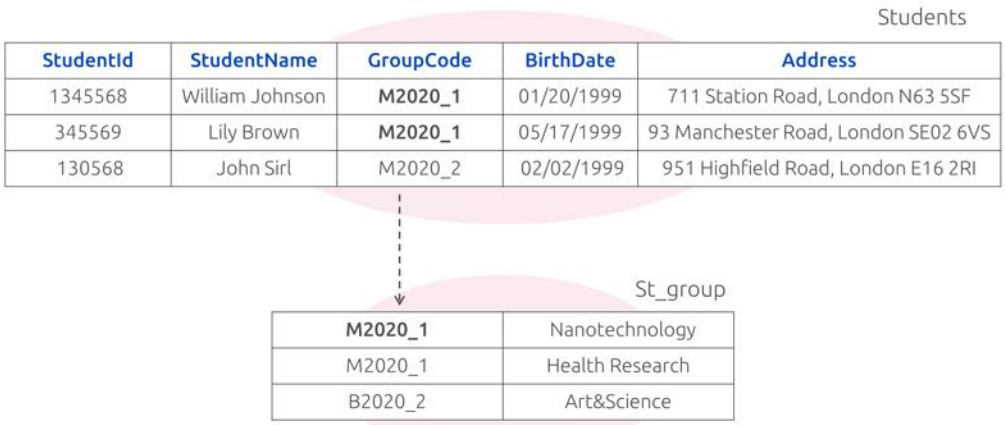
Ссылочная целостность

```
CREATE TABLE ST_GROUP  (
      GroupCode VARCHAR(32) PRIMARY KEY,
      Specialization VARCHAR(100) NOT NULL
   );

CREATE TABLE STUDENTS  (
      StudentId INTEGER PRIMARY KEY,
      StudentName VARCHAR(100) NOT NULL,
      GroupCode VARCHAR(32) REFERENCES ST_GROUP(GroupCode) ,
      BirthDate DATE,
      Address VARCHAR(100) NULL;
      UNIQUE (GroupCode, StudentName, BirthDate)
   );
```

Зададим для столбца с кодом группы в таблице `STUDENTS` внешний ключ – столбец `GroupCode` из таблицы `ST_GROUP`. Такие ограничения не позволят добавить в таблицу `STUDENTS` значения в поле GroupCode, отсутствующие в соответствующем столбце связанной таблицы. Таблицу `ST_GROUP` называют еще родительской.

Заметим, что внешний ключ должен быть такого же типа, как и связываемый столбец – в нашем случае они оба имеют тип `VARCHAR(32)`.



Кроме того, поле внешнего ключа должно быть объявлено либо уникальным значением, либо первичным ключом.

Если строки родительской таблицы `ST_GROUP` будут меняться или удаляться, то автоматически будет происходить поиск связанных строк в таблице `STUDENTS`. Предположим, что мы захотим удалить из таблицы `ST_GROUP` группу M2020_1 специальности Nanotechnology.

Тогда в зависимости от настроек ссылочной целостности, возможны несколько вариантов: например, можно запретить удаление этой группы, т.к. есть студенты, связанные с ней; либо можно удалить все связанные записи зависимых таблиц – при удалении группы удалять студентов, которые в ней учатся.

Последний вид ограничений, который мы рассмотрим, связан с указанием допустимого диапазона значений.

Ограничение допустимого диапазона значений

```
CREATE TABLE STUDENTS (
    StudentId INTEGER PRIMARY KEY CHECK (StudentId>0),
    StudentName VARCHAR(100) NOT NULL,
    GroupCode VARCHAR(32) REFERENCES ST_GROUP(GroupCode),
    BirthDate DATE,
    Address VARCHAR(100) NULL;
    UNIQUE (GroupCode, StudentName, BirthDate)
);
```

Это ограничение указывается ключевым словом `CHECK()` после описания атрибута. Если нужно проверить выполнение нескольких простых условий, то их можно соединять логическими операторами, например: `CHECK(StudentID>5 AND StudentID<500)`

При добавлении, удалении и изменении записей в таблицах будут автоматически проверяться все заданные ограничения целостности.

Таким образом ограничения целостности не позволят некорректным значениям попасть в базу данных.