**Лабораторная работа №1**

**Реляционная модель данных. Нормализация отношений.**

**Цель работы: Приобрести навыки проектирования и нормализации базы данных**

**Теоретические сведения:** материалы лекции 1,2

Наиболее распространенной на данный момент моделью данных, применяемой при проектировании баз данных, является **реляционная модель**. Суть этой модели состоит в представлении данных в виде двумерных таблиц, называемых **отношениями**.

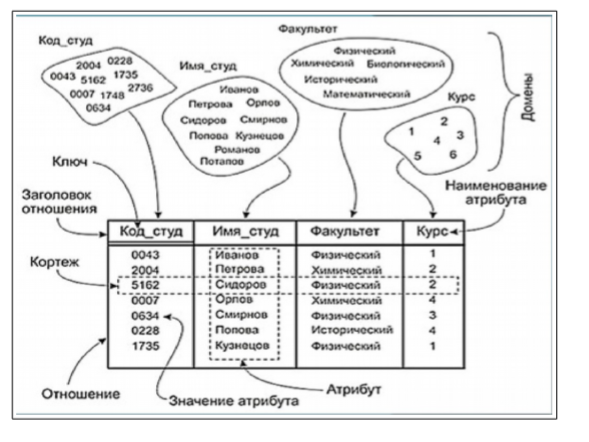


Рис. «Пример таблицы — отношения»

Отношение состоит из **строк (кортежей, записей)** и **столбцов (колонок, атрибутов, полей)**. Верхняя строка таблицы называется заголовком отношения. Каждый столбец в отношении должен иметь **уникальное имя**. Количество строк в отношении называется **кардинальным числом отношения**, а количество столбцов — **степенью отношения**. **Кортежи и атрибуты в отношении не упорядочены**. В отношениях не должно быть кортежей — дубликатов, по этой причине очень часто в отношение вводится такое поле, которое уникальным образом будет определять кортеж, это поле называется **первичным ключом** отношения. Из этого следует , что **единственный способ** **обращения к строке отношения** — **это** **обращение по ключу**, **а к столбцу — по имени**. Каждый атрибут относится к какому-либо **домену**, т. е. типу данных. Значения, содержащиеся в данном столбце ограничены этим типом данных. Например, на рисунке поля Код\_студ и Курс принадлежат к числовому домену, а Имя\_студ и Факультет к символьному. Информация в реляционных БД декомпозирована, т. е. хранится в различных таблицах, которые взаимосвязаны между собой. Связь между такими таблицами осуществляется с помощью пары **первичный ключ**-**внешний ключ**.

**Внешний ключ** — это столбец зависимой таблицы, каждое значение которого в текущем состоянии всегда входит во множество значений первичного ключа главной таблицы.

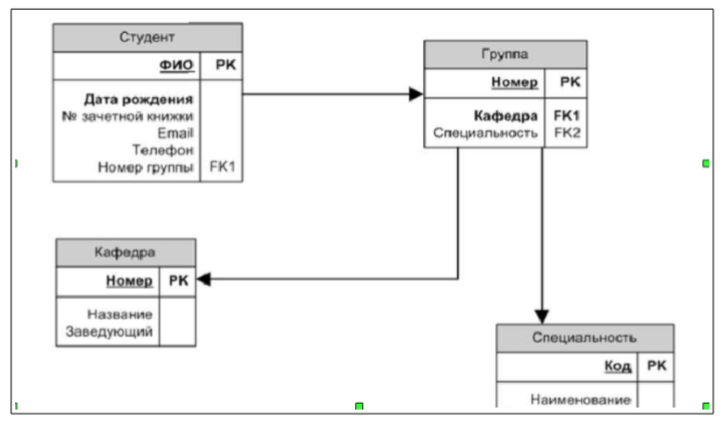


Рис. «Связь таблиц — отношений»

На рисунке аббревиатурой PK обозначен первичный ключ отношения, а FK — внешний. Так отношения Группа и Студент связаны парой Группа.Номер(PK) — Студент.Номер группы(FK). Отношения Группа и Специальность связаны парой Специальность.Код(PK) — Группа. Специальность(FK). Отношения Кафедра и Группа связаны Кафедра.Номер(PK) — Группа.Кафедра (FK). Если база данных спроектирована не очень хорошо, то при работе с ней будут накапливаться разного рода **аномалии** **данных**. Например, если данные о предметах и студентах будут храниться в одной таблице, то при удалении из таблицы записи о студенте, может быть удалена и информация о предмете как таковом. Или не будет возможности хранить информацию о предмете без информации о студенте.

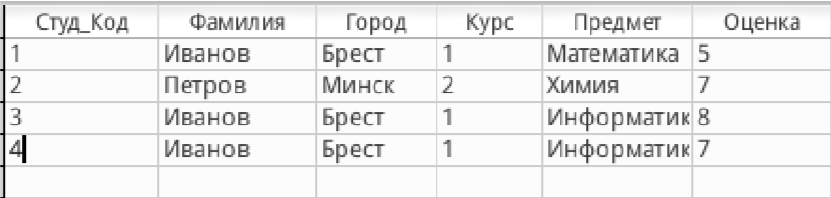


Рис. Таблица «Студенты»

Для исключения аномалий БД нормализуется с помощью системы шести нормальных форм. Каждая следующая нормальная форма, наследует требования предыдущей и добавляет к ним свои. Рассмотрим первые три из них, т. к. они дают достаточно мало шансов для возникновения аномалий данных.

**Первая нормальная форма**

• в каждой строке должны содержаться данные, соответствующие некоторому объекту или его части

• в каждой ячейке должно находиться только единственное значение

• у каждого столбца должно быть уникальное имя

• все строки должны быть различными

• порядок расположения строк и столбцов не имеет значения

Выше приведенная таблица находится в первой нормальной форме, однако она, как уже отмечалось, не избавлена от аномалий данных.

**Вторая нормальная форма**

Для того, чтобы таблица была во второй нормальной форме, необходимо, чтобы она была в первой нормальной форме и все ее неключевые атрибуты зависели от первичного ключа этого отношения. В рассмотренном примере первичным ключом отношения является поле Студ\_Код, при этом неключевое поле Предмет не зависит от первичного ключа, что и вызывает выше описанную аномалию. При приведении БД ко второй нормальной форме получатся отношения Студенты и Предметы:

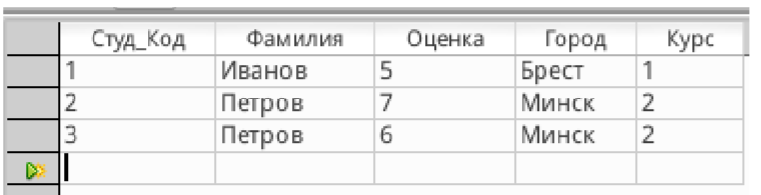


Рис. Таблица «Студенты»

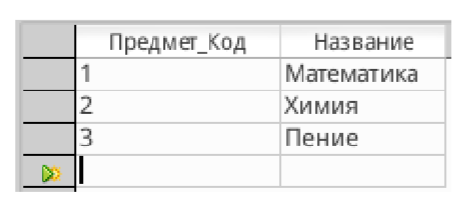


Рис. Таблица «Предметы»

**Третья нормальная форма**

Однако, после приведения ко второй нормальной форме БД не избавлена от аномалий даных, т. к. в таблице Студенты совместно хранятся персональные данные студента (Фамилия и т. д. …) и его оценки, что приводит к избыточности данных, а также, если удалить информацию об оценке, можно и удалить информацию о студенте. Избавиться от таких аномалий позволяет третья нормальная форма, которая требует от отношения наличия второй нормальной формы, а также , чтобы в нем не присутствовали транзитивные зависимости неключевых атрибутов от ключевых. В данном случае транзитивной является зависимость Оценка → Фамилия → Студ\_Код. Третья нормальная форма приведет к декомпозиции отношения Студенты на два отношения Студенты, где будет содержаться персональная информация студента, и Оценки, где будет содержаться информация об оценках студента.



Рис. «Студенты»

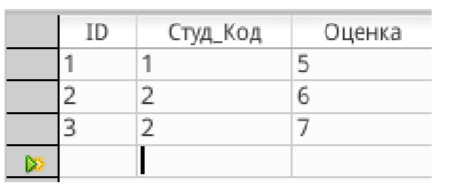


Рис. «Оценки»

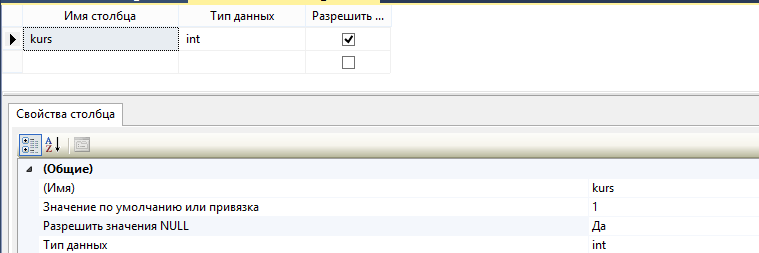
**Механизмы целостности данных**

Во время проектирования базы данных необходимо заботиться о целостности данных. Правильная структура таблиц позволяет защитить данные от нарушения связей и внесения неверных значений. Целостность данных основывается на стойкости и точности данных, которые хранит база данных. Существуют различные типы целостности данных:

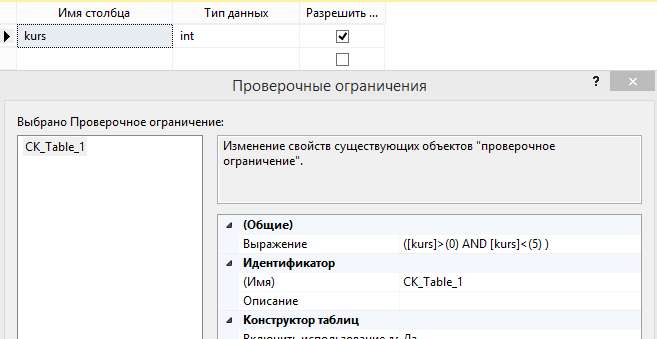
* Целостность полей – указывает набор значений данных, которые являются правильными для поля, и определяет, возможно ли использование нулевого значения. Например, поле для хранения пола человека может содержать одно из двух значений – М или Ж. Во-первых, этого достаточно, во-вторых, других значений пола просто не бывает, и необходимо запретить ввод других букв в данное поле. Целостность полей чаще всего (и лучше) обеспечивается с помощью ограничения CHECK, формата (с помощью шаблона) или региона возможных значений для поля.
* Целостность таблицы – требует, чтобы все строки в таблице имели уникальный идентификатор, называемый первичным ключом.
* Целостность ссылок – подразумевает отношения между первичным ключом (таблицы, на которую ссылаются) и внешним ключом (таблицы, которая ссылается на другую) всегда защищенными. Строка основной таблицы, на которую ссылаются записи из других таблиц, не может быть удалена, пока существуют ссылки на нее. Также не может быть изменено значение первичного ключа, пока на него существуют ссылки.

Стандарт SQL определяет следующие механизмы целостности данных:

* Default – значение по умолчанию позволяет подставлять в поле определенное значение при вставки, если иное не предусмотрено



* Check – ограничение на значения колонки позволяет контролировать ввод значений в столбец таблицы



* Первичный ключ – Внешний ключ. Пара обеспечивает согласованность данных в главной и зависимых таблицах

Ограничение PRIMARY KEY определяет первичный ключ таблицы, который уникально идентифицирует строку. Это гарантирует целостность таблицы.

* в таблицы может быть только один первичный ключ, но этот ключ может состоять из нескольких полей;
* первичный ключ не может содержать NULL - значение;
* для полей первичного ключа создается индекс.

Ограничение FOREIGN KEY (внешний ключ) гарантирует ссылочную целостность. Ограничение внешнего ключа определяет ссылку на колонку с первичным ключом или уникальную колонку в этой же или другой таблице. С помощью такого ключа обеспечивается целостность связей между таблицами. Внешний ключ, как и первичный, может состоять из нескольких полей. При создании связующего ключа, количество колонок внешнего ключа должно соответствовать количеству колонок первичного ключа, с которым происходит связь.

- Автоинкремент – механизм автоматической генерации уникальных значений в поле. Имеет начальное значение и приращение. В SQLServer автоинкрементация называется IDENTITY. Начальное значение и приращение у IDENTITY задается следующим образом: IDENTITY(1,2). В данном примере начальное значение равняется 1, а приращение 2.

**Типы данных SQLServer**

**Числовые типы данных**

**BIT**: хранит значение 0 или 1. Фактически является аналогом булевого типа в языках программирования. Занимает 1 байт.

**TINYINT**: хранит числа от 0 до 255. Занимает 1 байт. Хорошо подходит для хранения небольших чисел.

**SMALLINT**: хранит числа от –32 768 до 32 767. Занимает 2 байта

**INT**: хранит числа от –2 147 483 648 до 2 147 483 647. Занимает 4 байта. Наиболее используемый тип для хранения чисел.

**BIGINT**: хранит очень большие числа от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807, которые занимают в памяти 8 байт.

**DECIMAL**: хранит числа c фиксированной точностью. Занимает от 5 до 17 байт в зависимости от количества чисел после запятой. Данный тип может иметь два параметра **precision** и **scale**: DECIMAL(precision, scale). Параметр precision представляет максимальное количество цифр, которые может хранить число. Это значение должно находиться в диапазоне от 1 до 38. По умолчанию оно равно 18.

Параметр scale представляет максимальное количество цифр, которые может содержать число после запятой. Это значение должно находиться в диапазоне от 0 до значения параметра precision. По умолчанию оно равно 0.

**NUMERIC**: данный тип аналогичен типу DECIMAL.

**SMALLMONEY**: хранит дробные значения от -214 748.3648 до 214 748.3647. Предназначено для хранения денежных величин. Занимает 4 байта. Эквивалентен типу DECIMAL(10,4).

**MONEY**: хранит дробные значения от -922 337 203 685 477.5808 до 922 337 203 685 477.5807. Представляет денежные величины и занимает 8 байт. Эквивалентен типу DECIMAL(19,4).

**FLOAT**: хранит числа от –1.79E+308 до 1.79E+308. Занимает от 4 до 8 байт в зависимости от дробной части. Может иметь форму определения в виде FLOAT(n), где n представляет число бит, которые используются для хранения десятичной части числа (мантиссы). По умолчанию n = 53.

**REAL**: хранит числа от –340E+38 to 3.40E+38. Занимает 4 байта. Эквивалентен типу FLOAT(24).

**Типы Дата-Время**

**DATE**: хранит даты от 0001-01-01 (1 января 0001 года) до 9999-12-31 (31 декабря 9999 года). Занимает 3 байта.

**TIME**: хранит время в диапазоне от 00:00:00.0000000 до 23:59:59.9999999. Занимает от 3 до 5 байт. Может иметь форму TIME(n), где n представляет количество цифр от 0 до 7 в дробной части секунд.

**DATETIME**: хранит даты и время от 01/01/1753 до 31/12/9999. Занимает 8 байт.

**DATETIME2**: хранит даты и время в диапазоне от 01/01/0001 00:00:00.0000000 до 31/12/9999 23:59:59.9999999. Занимает от 6 до 8 байт в зависимости от точности времени. Может иметь форму DATETIME2(n), где n представляет количество цифр от 0 до 7 в дробной части секунд.

**SMALLDATETIME**: хранит даты и время в диапазоне от 01/01/1900 до 06/06/2079, то есть ближайшие даты. Занимает от 4 байта.

**DATETIMEOFFSET**: хранит даты и время в диапазоне от 0001-01-01 до 9999-12-31. Сохраняет детальную информацию о времени с точностью до 100 наносекунд. Занимает 10 байт.

Распространенные форматы дат:

yyyy-mm-dd - 2017-07-12

dd/mm/yyyy - 12/07/2017

mm-dd-yy - 07-12-17

Распространенные форматы времени:

hh:mi - 13:21

hh:mi am/pm - 1:21 pm

hh:mi:ss - 1:21:34

hh:mi:ss:mmm - 1:21:34:12

hh:mi:ss:nnnnnnn - 1:21:34:1234567

**Символьные типы данных**

**CHAR**: хранит строку длиной от 1 до 8 000 символов. На каждый символ выделяет по 1 байту. Не подходит для многих языков, так как хранит символы не в кодировке Unicode. Количество символов, которое может хранить столбец, передается в скобках. Например, для столбца с типом CHAR(10) будет выделено 10 байт. И если мы сохраним в столбце строку менее 10 символов, то она будет дополнена пробелами.

**VARCHAR**: хранит строку. На каждый символ выделяется 1 байт. Можно указать конкретную длину для столбца - от 1 до 8 000 символов, например, VARCHAR(10). Если строка должна иметь больше 8000 символов, то задается размер MAX, а на хранение строки может выделяться до 2 Гб: VARCHAR(MAX).

Не подходит для многих языков, так как хранит символы не в кодировке Unicode.

В отличие от типа CHAR если в столбец с типом VARCHAR(10) будет сохранена строка в 5 символов, то в столбце будет сохранено именно пять символов.

**NCHAR**: хранит строку в кодировке Unicode длиной от 1 до 4 000 символов. На каждый символ выделяется 2 байта. Например, NCHAR(15)

**NVARCHAR**: хранит строку в кодировке Unicode. На каждый символ выделяется 2 байта.Можно задать конкретный размер от 1 до 4 000 символов: . Если строка должна иметь больше 4000 символов, то задается размер MAX, а на хранение строки может выделяться до 2 Гб.

Еще два типа **TEXT** и **NTEXT** являются устаревшими и поэтому их не рекомендуется использовать. Вместо них применяются VARCHAR и NVARCHAR соответственно.

**Бинарные типы данных**

**BINARY**: хранит бинарные данные в виде последовательности от 1 до 8 000 байт.

**VARBINARY**: хранит бинарные данные в виде последовательности от 1 до 8 000 байт, либо до 2^31–1 байт при использовании значения MAX (VARBINARY(MAX)).

Еще один бинарный тип - тип IMAGE является устаревшим, и вместо него рекомендуется применять тип VARBINARY.

**Прочие типы данных**

**UNIQUEIDENTIFIER**: уникальный идентификатор GUID (по сути строка с уникальным значением), который занимает 16 байт.

**TIMESTAMP**: некоторое число, которое хранит номер версии строки в таблице. Занимает 8 байт.

**CURSOR**: представляет набор строк.

**HIERARCHYID**: представляет позицию в иерархии.

**SQL\_VARIANT**: может хранить данные любого другого типа данных T-SQL.

**XML**: хранит документы XML или фрагменты документов XML. Занимает в памяти до 2 Гб.

**TABLE**: представляет определение таблицы.

**GEOGRAPHY**: хранит географические данные, такие как широта и долгота.

**GEOMETRY**: хранит координаты местонахождения на плоскости.

**Ход работы**

1. Придумать и описать *свой* пример отношений, не находящихся в 1НФ, 2НФ, 3НФ. Привести способ их нормализации.
2. В соответствии с вариантом предметной области спроектировать физическую структуру БД, определив таблицы. Для каждой таблицы определить ее структуру. Определить связи между таблицами.
3. Обеспечить нормализацию базы данных до 3НФ
4. Для полей таблиц определить соответствующие механизмы целостности данных
5. Связи между таблицами представить через диаграмму связей
6. Составить отчет

**Варианты заданий:**

1. **База данных медицинского кооператива**

База данных используется автоматизированной системой поликлиники. В базе данных должно быть предусмотрено хранение имени, пола, даты рождения, домашнего адреса каждого пациента. Информация о визите пациента к врачу в поликлинику, а также информация о визите врача на дом к пациенту заносится в базу данных: дата и место приема, симптомы, диагноз и предписания больному, персональные данные пациента и врача. Если врач прописывает больному лекарство, то информация о нем, о способе его приема, возможных побочных эффектах заносится в базу данных.

1. **База данных аптеки**

База данных хранит информацию об аптекарях-кассирах, работающих в аптеке; о лекарствах с указанием их названий, производителе, аналогах, о партии поступления, о дате поступления, количестве поступления; о продаже лекарств с указанием даты продажи, аптекаря-кассира, цене, количестве, стоимости

1. **База данных законодательной палаты**

В базе данных хранятся имена, адреса, домашние и служебные телефоны всех законодателей. В законодательной палате работает определенное число комиссий, все участники которых являются членами законодательной палаты. Каждая комиссия имеет свой профиль (образование, медицина, жилье и т.д.) Данные по каждой из комиссий включают: председатель и состав комиссии. Члены палаты могут заседать в нескольких комиссиях. В базу заносятся время и место проведения заседания, с указанием ее членов, принявших участие в работе комиссии

1. **База данных ипподрома**

В информационной системе ипподрома должны храниться данные об участвующих в скачках лошадях (кличка, пол, возраст), их владельцах (имя, адрес, телефон) и жокеях (имя, адрес, возраст, рейтинг). Необходимо сформировать таблицы для хранения информации по каждому состязанию: дата, время и место проведения, название состязаний, номера заездов, клички участвующих в заездах лошадей и имена жокеев, занятые ими места и показанное в заезде время.