ЦИФРОВАЯ КУЛЬТУРА
УНИВЕРСИТЕТ ИТМО

Лекция **«Реляционные базы данных»**

Михайлова Елена Георгиевна
Графеева Наталья Генриховна

Санкт-Петербург
2019

Содержание

1 Системы управления базами данных

В современном мире нас окружают различные информационные системы. Благодаря им различные задачи, еще недавно требовавшие значительное время на выполнение, становятся простыми и незаметными. С помощью информационных систем мы общаемся в социальных сетях, совершаем покупки, записываемся на прием к врачу. В основе любой информационной системы

лежит база данных, в которой накапливается вся необходимая информация. Когда мы говорили о первичной обработке данных, то в качестве инструмента мы пользовались электронными таблицами. Но когда данных становится много, и они имеют сложную структуру, инструменты нужны совсем другие.

База данных представляет собой организованную совокупность данных, которые, как правило, хранятся и обрабатываются в электронном виде. Объемы накопленных данных могут достигать огромных размеров. Но с этими объемами легче справляться, когда данные имеют четкую структуру, в них можно выделить типовые объекты, их элементы, или составные части, проследить связи между объектами. Такие данные называют структурированными. Часто данные не имеют четкой определенной единообразной структуры, являясь просто набором разнородных документов. Такие данные называются неструктурированными, и работать с ними гораздо сложнее. Полуструктурированными, или слабоструктурированными, называют данные, если в них можно выделить хотя бы некоторые элементы структуры.

Конечно, любая ИС кроме данных содержит ПО, которое помогает решать задачи предметной области. Но в любой ИС нужно иметь возможность описывать структуру данных, сохранять, изменять и быстро находить данные, позволять работать с ними одновременно нескольким пользователям и заботиться о том, чтобы данные не пропали и не были повреждены. Для этих операций логично выделить специальное ПО, которое называется системой управления базой данных.

1.1 Что же такое СУБД?

СУБД можно определить как совокупность средств и инструментов для управления базой данных и организации взаимодействия с ней. Система управления базами данных (СУБД) — это программное обеспечение, которое взаимодействует с конечными пользователями, приложениями и самой базой данных для сбора, хранения, обработки и анализа данных. Возможности, которые предоставляет СУБД пользователю:

1. Постоянное хранилище. Как и файловая система, СУБД поддерживает хранение очень больших объемов данных, которые существуют независимо от любых процессов, использующих данные. СУБД выходит далеко за рамки файловой системы в хранении структур данных, которые поддерживают эффективный доступ к очень большим объемам информации.

2. Программный интерфейс. СУБД позволяет пользователю или прикладной программе получать доступ к данным и изменять их с помощью мощного языка запросов высокого уровня. Опять же, преимущество СУБД перед файловой системой заключается в гибкости управления сохраненными данными гораздо более сложными способами, чем чтение и запись файлов.

3. Управление транзакциями. СУБД поддерживает параллельный доступ к данным, т. е. одновременный доступ сразу к нескольким различным процессам (называемым «транзакциями»). Чтобы избежать некоторых нежелательных последствий одновременного доступа, СУБД поддерживает изоляцию, которая создает видимость, что транзакции выполняются по одиночке, и атомарность, требование, чтобы транзакции выполнялись полностью или не выполнялись вообще. СУБД также поддерживает долговечность, которая заключается в том, что данные от завершенных транзакций не могут быть потеряны, и возможность восстановления после сбоев или ошибок различных типов.

Современные СУБД принято делить на два класса:

- реляционные;
- NOSQL.

Хранилища NOSQL предназначены для хранения неструктурированных или слабоструктурированных данных.

Для хранения структурированных данных наибольшее распространение получили реляционные базы данных, когда хранимая информация может быть представлена в табличном виде. Конечно, реляционные базы данных — это не просто плоские таблицы — данные разных таблиц могут быть связаны между собой, иметь некоторые ограничения целостности и т.д.

Реляционные СУБД берут начало с 70-х годов прошлого века, когда появился первый исследовательский прототип реляционной СУБД, который разрабатывался в IBM под названием System Relational, или System R. System R стала первой реализацией Structured Query Language (SQL), который является теперь стандартом для реляционных БД. С тех пор появлялось много СУБД, основанных на реляционной модели.

Несмотря на то, что все системы управления базами данных выполняют одни и те же основные задачи, их функции и возможности могут существенно отличаться.

Наиболее важные параметры — это производительность и надежность работы. И, конечно, цена имеет значение — Стоимость коммерческих версий некоторых систем исчисляется десятками или сотнями тысяч долларов. Но почти у всех коммерческих систем существуют свободно распространяемые версии, обычно с неполным функционалом и работающих на ограниченном объеме данных, но все же на них можно разрабатывать небольшие коммерческие системы.

При сравнении различных популярных баз данных, следует учитывать, удобна ли для пользователя и масштабируема ли данная конкретная СУБД, а также необходимо убедиться, что она будет хорошо интегрироваться с другими продуктами, которые уже используются в системе для решения прикладных задач. Различные СУБД документированы по-разному: более или менее тщательно. По-разному предоставляется и техническая поддержка. Также стоит обратить внимание на возможности процедурного расширения языка SQL и поддерживаемый набор пользовательских интерфейсов, которые позволят разрабатывать приложения.

Основные отличия СУБД

- производительность и надежность работы;
- коммерческое ПО или свободно распространяемое;
- возможность интеграции с другими системами;

- документация и техническая поддержка;
- возможности встроенных функций;
- поддерживаемый набор пользовательских интерфейсов.

1.2 Наиболее популярные реляционные СУБД

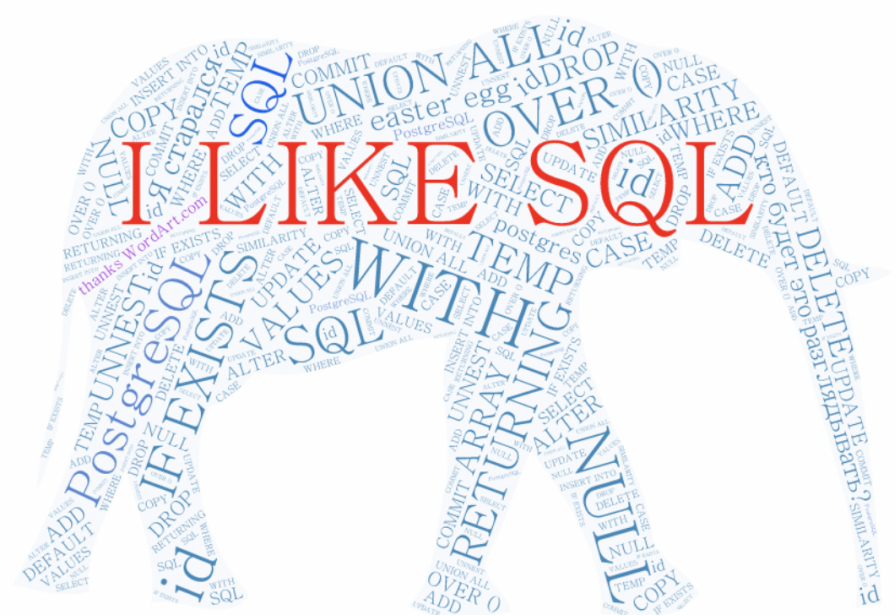
Самая популярная реляционная СУБД в настоящее время — это Oracle. Компания ORACLE основана в 1977 году. Разрабатываемая этой компанией СУБД Oracle считается первой коммерческой СУБД с поддержкой языка запросов SQL, и одной из первых реляционных СУБД. Согласно ряда рейтингов, Oracle занимает лидирующие позиции для разработки баз данных крупных корпораций.

Большую популярность на рынке также имеет Microsoft SQL Server, ориентированный, в основном, на операционную систему Windows.

IBM выпускает СУБД под названием под названием DB2, которая является коммерческим продолжением СУБД System R.

Кроме коммерческих СУБД существуют решения с открытым программным кодом, к которым относятся SQLite, PostgreSQL и различные версии MySQL, например, MariaDB. Лидером среди них является PostgreSQL благодаря поддержке сложных структур и широкого спектра встроенных функций и определяемых пользователем типов данных.

Несмотря на то, что систем много и у них есть некоторые отличия, если хорошо освоить работу с одной из них, перейти к использованию другой будет не трудно.



2 Структурирование данных

Проектирование базы данных надо начинать с анализа предметной области и выявления требований к ней отдельных пользователей. У разных пользователей базы данных могут быть разные задачи, и свой набор данных, необходимый для их решения. Обобщенное представление пользователей называют схемой базы данных. С чего начать составление схемы?

Сначала необходимо создать обобщенное неформальное описание. Это описание делают с использованием естественного языка, математических формул и других средств, понятных всем людям, работающих над проектированием данных. Давайте рассмотрим базу данных по продажам товара. Нужно хранить информацию о товарах — название, цена и артикул, о покупателях — ФИО, адрес и о заказах — кто что заказал, когда и в каком количестве. Чтобы не усложнять совсем нашу схему, будем считать, что каждый товар заказывается отдельно, т.е. в одном заказе будет только один вид товара.

Следующий шаг — выделить структуру данных. Необходимо определить объекты, информацию о которых будем хранить, выделить их свойства, называемые атрибутами, и описать связи, в которые объекты могут вступать друг с другом.

Чтобы описать данные более формально и наглядно, часто используют диаграмму «Сущность-связь», или ER-диаграмму, которая дает графическое представление логической структуры данных.

Базовым понятием при построении ER-диаграммы является сущность. Сущность — это некоторый объект, реальный или воображаемый, часть окружающего мира, событие, информация о котором важна в рамках нашей предметной области. Как правило, каждая предметная область содержит в себе множество сущностей. У каждой сущности должно быть уникальное имя. В нашем примере сущностями будут товары и покупатели. Сущности на схеме изображаем с помощью прямоугольников.

Таким образом сущность описывает класс или множество объектов, соответствующих определенному типу. Экземпляром сущности называют конкретный объект некоторого типа. Например, у сущности Товар экземплярами могут быть Ручка, Карандаш, Тетрадка.

Отдельные характеристики, или свойства сущностей называются атрибутами.

Каждый атрибут также имеет уникальное имя (в рамках данной сущности) и у каждого атрибута должен быть определен тип данных. Например, у сущности Товар атрибутами будут Артикул, название и цена. У сущности Покупатель атрибутами будут Фамилия, Имя, Отчество, Адрес. На схеме атрибуты окружаем овалами и связываем их с сущностями ненаправленными

дугами.

Нам нужно отличать друг от друга экземпляры сущностей каждого вида. Если у объекта есть один (или несколько атрибутов), которые позволяют его однозначно идентифицировать, то такие атрибуты являются ключом. Ключ — очень важное понятие для проектирования данных. Для сущности Товар таким ключом может быть Артикул.

Если не удается найти естественный ключ — например, среди покупателей могут оказаться тезки, и мы даже не можем исключить возможность, что они проживают по одному адресу. Тогда для удобства идентификации можно использовать искусственный, или суррогатный ключ –Код покупателя.

Ключевой атрибут на схеме обычно подчеркивают сплошной линией. Иногда ключ может быть составным, то есть состоять из нескольких атрибутов — например, если бы мы захотели хранить паспортные данные покупателей, то появился бы составной ключ — Серия + Номер паспорта

Бывает, что у сущности может быть несколько ключей — в нашем случае в роли ключа могут выступать либо Код покупателя, либо Серия + Номер паспорта.

Если у сущности несколько ключей, то из них надо выбрать наиболее важный ключ, по которому чаще происходит поиск. Такой ключ объявляют первичным ключом.

Обычно сущности взаимодействуют друг с другом. Чтобы указать это, на диаграммах изображают связи. Связи — это отношения между сущностями. В отличие от сущностей, связи не могут быть сами по себе, без указания на сущности, которые в связи участвуют. Например, если покупатель заказал определенный товар, то появляется связь Заказ.

Связи могут иметь собственные атрибуты. Например, у связи Заказ могут быть Атрибут Количество товара, дата заказа. Для идентификации связей используют ключи связываемых сущностей и, возможно, выделенные атрибуты связи. Каждой связи на схеме тоже желательно дать название. На схеме связи изображают в ромбах

2.1 Типы связей

В связи могут участвовать несколько сущностей. Мы рассмотрели бинарную связь между сущностями «Товар» и «Покупатель».

Если бы мы хотели бы хранить информацию о магазинах, в которых товар был приобретен, то связь была бы тернарной. Но такие связи трудно хранить, и их стараются не использовать.

Бывают рекурсивные связи, когда объекты одного вида ссылаются на такие же — например, при помощи рекурсии обычно реализуют иерархию: «Сотрудник -> Руководит».

Рассмотрим бинарные связи. Они делятся на три вида в зависимости от

количества участвующих в них объектов.

Один к одному	1:1
Один ко многим	1:N
Многие ко многим	M:N

Количество объектов определяет степень связи. Если много объектов одного вида вступают в связь со многими объектами другого вида, то получается связь вида многие-ко-многим. Степень конца связи указывается графически, множественность связи изображается в виде «вилки» на конце связи. В нашем примере каждый покупатель может заказать несколько товаров, и каждый товар может быть куплен несколькими покупателями.

Если много объектов одного вида вступают в связь только в одном объектом другого вида, то получается связь вида многие-к-одному. Такая связь изображается прямой линией. Например, Товары продаются в Магазине. В каждом магазине может быть много товаров, но каждый товар находится ровно в одном магазине.

Заметим, что между объектами может быть несколько разных связей. Представим, что каждому покупателю надо выбрать Любимый товар — тогда для каждого покупателя он будет ровно один. Но один и тот же товар может быть любимым у нескольких покупателей, получается связь вида многие к одному. Например, в нашей схеме между объектами Товар и Покупатель получилось две связи — Заказ и Любимый товар.

А если один объект одного вида вступают в связь только с одним объектом другого вида, то получается связь вида один-к-одному. Например, добавим в нашу схему новый объект — директор магазина. У каждого магазина ровно один директор, и каждый директор руководит ровно одним магазином.

Осталось упомянуть модальность связей. Она служит для того, чтобы указать, является ли связь обязательной или возможной (необязательной).

Необязательная связь означает, что объект одного вида может быть связан с одним или несколькими экземплярами объектов другого вида, а может быть и не связан ни с одним экземпляром. Модальность связи также изображается графически — необязательность связи помечается кружком на конце связи. Например, могут быть товары, которые никто еще не заказывал. Тогда связь от Товара к Покупателю будет возможной.

Обязательные связи означают, что объект обязан быть связан не менее чем с одним экземпляром объектов другого вида. Например, каждый покупатель должен заказать хотя бы один товар.

Обязательность связи изображается прямой линией. Итак, мы научились описывать данные в терминах модели сущность связь, которая позволяет понять суть разрабатываемой базы данных. Теперь надо понять, как же преобразовать ER-диаграммы в структуры хранения.

3 Создание таблиц

После составления ER-диаграммы нужно подобрать СУБД, в которой будет реализована база данных, и преобразовать описанную схему в конкретные структуры хранения.

Наибольшее распространение для структурированных данных получила реляционная модель. В 1970 году Эдгар Кодд написал статью, в которой впервые описал реляционную модель данных. Основная мысль этой статьи состоит в том, что все данные, которые видят пользователи, могут быть представлены в виде таблиц. Говоря чуть более формально, в основе реляционной модели данных лежит понятие отношения, которое задается списком своих элементов и перечислением их значений. Реляционная модель данных представляет информацию в виде совокупности взаимосвязанных таблиц, которые принято называть отношениями. Таблицы состоят из столбцов и строк. Столбцы соответствуют атрибутам объектов, а строки — отдельным экземплярам. На пересечении строки и столбца находится конкретное значение атрибута, которое принадлежит некоторой области определения.

Как перейти из модели объект-связь к реляционной? В ER-модели мы описывали информацию о сущностях и связях между ними. С сущностями, или объектами все просто — каждый объект превращается в отдельную таблицу. Имя объекта становится именем таблицы, и это имя должно быть уникально в рамках одной базы данных. Атрибуты становятся названиями столбцов.

Заголовок таблицы состоит из имен атрибутов Каждый столбец соответствуют одному атрибуту, и для каждого столбца задается свой тип данных. Каждая строка таблицы соответствует отдельному экземпляру объекта. Ключом таблицы становится ключ сущности. Как в реляционной модели представлены связи? Они также хранятся в отношении.

Схема данного отношения составляется из ключевых атрибутов объектов, участвующих в связи. Для связи вида 1 к одному можно объединить в одной таблице столбцы, соответствующие атрибутам обоих объектов. В качестве ключа отношения можно выбрать ключ от одного из объектов.

Для реализации связи «Руководит магазином» получается одна таблица, в которой хранится информация и о магазине, и о его директоре.

Если много объектов одного вида вступают в связь только с одним объектом другого вида, то имеем связь вида многие-к-одному. Примером такой связи является связь «Любимый продукт» между Покупателем и Товаром. Для представления объектов Покупатель и Товар понадобятся две таблицы.

Для реализации связи надо добавить ключ таблицы Товар в таблицу Покупатель.

А вот пример таблицы Покупатель, в котором уже появился столбец Код

товара с указанием на любимый товар. Ключом таблицы при этом останется Код покупателя.

Осталось реализовать самую сложную, но наиболее часто встречающуюся связь — многие-ко-многим. Между Покупателем и Товаром как раз такая связь Заказ.

Для реализации связи вида многие ко многим потребуется создать отдельную таблицу, столбцами которой будут ключевые атрибуты связанных объектов. Возьмем столбцы Код покупателя, Артикул товара. Назовем новую таблицу Order. Не забудем добавить собственные атрибут связи — количество товара и дату заказа. Итак, любые объекты и связи реального мира могут быть представлены в виде таблиц.

4 SQL DDL

Чтобы создать таблицу, нам нужно использовать команды. В реляционных СУБД для этого есть язык SQL — это широко распространенный и стандартизированный язык программирования.

Все команды языка SQL условно делятся на несколько групп в зависимости от их назначения:

1. Команды определения данных (Data Definition Language, DDL), которые позволяют создавать саму базу данных, таблицы и другие необходимые объекты.

2. Команды манипуляции данными (Data Manipulation Language, DML) — при помощи этих команд мы можем заполнять таблицы данными, изменять их или удалять. И, конечно, находить данные по заданным критериям.

3. Команды определения доступа к данным (Data Control Language, DCL). Эти операторы нужны, чтобы предоставлять пользователям доступ к базе данных.

4. Команды управления транзакциями (Transaction Control Language, TCL). Механизм транзакций нужен для того, чтобы обеспечить возможность одновременной работы нескольких пользователей так, чтобы они не мешали друг другу. Также транзакции позволяют восстановить корректное состояние базы данных после возможных сбоев.

5. Команды определения данных (Data Definition Language, DDL).

Первое, что необходимо создать — это сама база данных, т.е. область памяти, в которой хранятся все объекты базы данных. База данных создается командой CREATE.

В некоторых СУБД существует еще один важный объект — схема (SCHEMA), который используется как контейнер, содержащий объекты базы данных. В этом случае база данных рассматривается как множество схем.

При создании базы данных необходимо указать ее имя. Дополнительно можно указать множество параметров, как место хранения и начальный размер хранения, коэффициент прироста и т.д.

После создания базы данных можно приступать к созданию таблиц. Таблицы в базе данных также создаются с помощью команды CREATE. Необходимо задать имя таблицы и дать описание столбцов.

Для определения каждого столбца таблицы необходимо указать его название и тип данных. Желательно, чтобы название столбца передавало логику организации данных. Например, столбец, соответствующий Коду товара может называться `Id`, `Product_id`.

Для хранения данных основными типами являются текст, числа и время. Также есть различные временные форматы и некоторые специальные, дополнительные для хранения данных в формате XML, денежных, битовых и пространственных данных, для организации иерархической структуры и пр. Описание типов может немного отличаться в различных СУБД, мы будем приводить примеры для PostgreSQL.

Числа можно хранить целые или вещественные. Например, в таблице Товар есть атрибут Артикул — это целое число. Для него подойдет тип `integer`. А для поля Цена число должно быть вещественный с точностью до сотых. `numeric(10,2)` Параметрами поля является общее количество цифр до и после запятой и количество знаков после десятичного разделителя. Поле Названия товара — строка. Строки могут быть фиксированной длины — задавая длину, мы ограничиваем максимальный размер строки, например, `char(50)`. Если строка окажется короче 50 символов, то она будет дополнена пробелами, а если длиннее, то в значение столбца попадут только первые 50 символов, остальные будут обрезаны. Если длина атрибута может меняться в зависимости от содержимого, то можно использовать поля переменной длины, `varchar(n)`. Тогда поле не будет дополняться пробелами, а вместе со значением строки будет храниться ее длина.

```
CREATE TABLE Products (
Product_ID integer,
PRODUCT_NAME varchar(25),
PRICE numeric(10,2)
)
```

Таким образом мы описали столбцы таблицы Товар.

Дополнительно можно указать ограничения, которым должны удовлетворять данные. Назначение ограничений — не допустить попадания в базу данных некорректных данных.

По умолчанию (если не сказано иначе) поля таблицы могут иметь незаданное значение. Но при незаданных значениях некоторых полей элемент данных не имеет смысла. Начнем с того, что у Товара обязательно должно быть название и артикул. А полю цена позволим иметь незаданные значения.

```
CREATE TABLE Products (
Product_ID integer NOT NULL,
PRODUCT_NAME text NOT NULL,
PRICE numeric(10,2) NULL
)
```

Очень важное ограничение — первичный ключ таблицы. Он не может иметь незаданных значений и должен быть уникален для каждой строки. Первичный ключ в таблице может быть только один.

```
CREATE TABLE Products (
Product_ID integer PRIMARY KEY,
PRODUCT_NAME text NOT NULL,
PRICE numeric(10,2) NULL
)
```

Если нам понадобится сделать другие поля таблицы уникальными, то можно добавить `UNIQUE`.Если бы мы захотели, чтобы все товары имели уникальное название, то описание таблицы выглядело бы так:

```
CREATE TABLE Products (
Product_ID integer PRIMARY KEY,
PRODUCT_NAME text UNIQUE NOT NULL,
PRICE numeric(10,2) NULL
)
```

Уникальными можно объявить комбинацию из нескольких полей. Например, можем сказать, что не может быть двух товаров с одинаковым названием и одинаковой ценой.

```
CREATE TABLE Products (
Product_ID integer PRIMARY KEY,
PRODUCT_NAME text NOT NULL,
PRICE numeric(10,2) NULL,
UNIQUE(PRODUCT_NAME, PRICE)
)
```

Можно добавить проверку некоторых условий, которым должны удовлетворять поля. Например, укажем, что цена товара всегда положительна.

```
CREATE TABLE Products (
Product_ID integer PRIMARY KEY,
PRODUCT_NAME text NOT NULL,
PRICE numeric(10,2) NULL CHECK(PRICE>0)
)
```

Теперь можем создать еще одну таблицу Покупатель.

```
CREATE TABLE Customers (
Customer_ID integer PRIMARY KEY,
Cust_Last_NAME text NOT NULL,
Cust_First_NAME text NOT NULL,
Cust_Address text NULL
)
```

Осталось создать таблицу Продажа, в которой будет информация о покупателе, товаре и количестве.

```
CREATE TABLE Orders (
Order_ID integer PRIMARY KEY,
Customer_ID integer,
Product_ID integer,
Quantity integer,
Order_date date
)
```

Эта таблица нужна для реализации связи между покупателем и товаром. Но что будет, если мы ошибемся в значении поля `Product_ID`? Например, попробуем записать информацию о покупке товара с кодом, которого не существует в таблице Товар? В таблице товаров такого товара мы не найдем.

Для целостности данных нужно разрешать указывать в таблице Покупка только те коды товара, которые есть в таблице Товар. В таком случае нам поможет ограничение внешний ключ. Мы свяжем поля таблиц `Orders` и `Products` и тем самым запретим помещать в таблицу некорректные значения.

```
CREATE TABLE Orders (
Order_ID integer PRIMARY KEY,
Customer_ID integer,
Product_ID integer REFERENCES Product(Product_ID),
Quantity integer,
Order_date date
)
```

Аналогично свяжем таблицу `Orders` с таблицей `Customer`:

```
CREATE TABLE Orders (
Order_ID integer PRIMARY KEY,
Customer_ID integer REFERENCES Customers (Customer_ID),
Product_ID integer REFERENCES Product(Product_ID),
Quantity integer,
Order_date date
)
```

Ограничение внешний ключ используется для связывания полей из разных таблиц. Связываемые поля должны совпадать по типу, и поле, на которое организуется ссылка, должно иметь уникальные значения или быть первичным ключом.

Мы научились создавать таблицы. Если таблица оказалась не нужна, то ее можно удалить командой `DROP TABLE` — будет удалены все данные, если они были в таблице, и структура таблицы.

Командой `ALTER TABLE` можно изменить структуру таблицы, добавить или удалить столбцы и ограничения целостности. Например, изменим столбец `Customer_ID` таблицы `Orders` — добавим ограничение на отсутствие незаданных значений.

```
ALTER TABLE Orders MODIFY Customer_ID integer not null;
```

5 SELECT

Мы научились создавать таблицы, теперь надо заполнить их данными. Для этого нам поможет оператор `INSERT`. Вставим строку в таблицу Товаров:

```
INSERT INTO PRODUCTS VALUES(8, 'Карандаш', 0.3)
```

Если нужно изменить запись, воспользуемся командой `UPDATE`

```
UPDATE PRODUCTS SET Product_Name='Карандаш простой'
WHERE Product_ID=8
```

При вставке и изменении строк проверяются все ограничения, которые мы указывали при описании таблицы. Например, при попытке добавить в таблицу Товаров следующую запись будет ошибка, т.к. товар с артикулом 1 уже существует, а поле артикул должно содержать только уникальные значения, т.к. объявлено первичным ключом.

```
INSERT INTO PRODUCTS VALUES(8, 'Фломастер', 0.3)
Для удаления записи есть команда DELETE
DELETE FROM PRODUCTS WHERE Product_ID=8
```

С этой командой надо быть осторожным. Если запустить ее без параметров, то все строки таблицы будут удалены.

```
DELETE FROM PRODUCTS
```

Для поиска данных в таблицах есть всего одна команда, или инструкция `SELECT`, которая возвращает строки из таблиц базы данных согласно заданным критериям. `SELECT` позволяет делать выборку одной или нескольких строк или столбцов из одной или нескольких таблиц.

Самый простой запрос — вывести все строки таблицы.

```
SELECT * FROM PRODUCTS
```

Product_ID	Product_Name	Price
1	Табак листовой	0.20
2	Табак резаный	0.50
3	Сигары "Шик"	
4	Папиросы "Ира"	0.80
5	Папиросы "Пушка"	0.40
6	Тетрадка в клетку 12 л	0.20
7	Ручка шариковая	

Обработка команды `SELECT` начинается с раздела `FROM` — после него указывают, из каких таблиц будут выбираться данные. После слова `SELECT` указываются столбцы. `*` означает, что выводить нужно все столбцы таблицы Если нужно выводить некоторые столбцы, или изменить их порядок, то надо указать их имена в явном виде.

```
SELECT Product_ID, Product_Name FROM PRODUCTS
```

Product_ID	Product_Name
1	Табак листовой
2	Табак резаный
3	Сигары "Шик"
4	Папиросы "Ира"
5	Папиросы "Пушка"
6	Тетрадка в клетку 12 л
7	Ручка шариковая

Выводимые столбцы можно сразу переименовать

```
SELECT Product_ID ID, Product_Name Name FROM PRODUCTS
```

ID	Name
1	Табак листовой
2	Табак резаный
3	Сигары "Шик"
4	Папиросы "Ира"
5	Папиросы "Пушка"
6	Тетрадка в клетку 12 л
7	Ручка шариковая

Следующий запрос выводит все столбец с ценами из таблицы продуктов

```
SELECT PRICE FROM PRODUCTS
```

Price
0.20
0.50
-
0.80
0.40
0.20
-

Мы видим, что два раза выводится одинаковая цена — 0.2, т.к. цены на два товара совпадают.

Если нужно выводить только уникальные строки, то надо использовать ключевое слово `DISTINCT`

```
SELECT DISTINCT PRICE FROM PRODUCTS
```

Price
0.20
0.50
-
0.80
0.40
-

Используя имена столбцов и константы, можно строить выражения. Например, сосчитаем для каждого товара цену за 100 штук.

```
SELECT Product_ID, Product_Name, Price*100 as Price_Per_100
FROM PRODUCTS
```

Product_ID	Product_Name	Price_Per_100
1	Табак листовой	20
2	Табак резаный	50
3	Сигары "Шик"	
4	Папиросы "Ира"	80
5	Папиросы "Пушка"	40
6	Тетрадка в клетку 12 л	20
7	Ручка шариковая	

Если выборку надо отсортировать, то используем ключевое слово `ORDER BY`, после которого указываем критерии сортировки. По умолчанию сортировка производится по возрастанию.

```
SELECT * FROM PRODUCTS ORDER BY Price
```

Product_ID	Product_Name	Price
1	Табак листовой	0.20
6	Тетрадка в клетку 12 л	0.20
5	Папиросы "Пушка"	0.40
2	Табак резаный	0.50
4	Папиросы "Ира"	0.80
3	Сигары "Шик"	
7	Ручка шариковая	

Вместо имени столбца можно указать его порядковый номер в формируемой выборке. Столбец `Price` третий по счету, тогда можно написать

```
SELECT * FROM PRODUCTS ORDER BY 3
```

Product_ID	Product_Name	Price
1	Табак листовой	0.20
6	Тетрадка в клетку 12 л	0.20
5	Папиросы "Пушка"	0.40
2	Табак резаный	0.50
4	Папиросы "Ира"	0.80
3	Сигары "Шик"	
7	Ручка шариковая	

Если надо сортировать по убыванию, то после критерия сортировки используют ключевое слово `DESC`:

```
SELECT * FROM PRODUCTS ORDER BY 3 DESC
```

Product_ID	Product_Name	Price
4	Папиросы "Ира"	0.80
2	Табак резаный	0.50
5	Папиросы "Пушка"	0.40
1	Табак листовой	0.20
6	Тетрадка в клетку 12 л	0.20
3	Сигары "Шик"	
7	Ручка шариковая	

Можно добавить еще критерий сортировки — например, те строки, у которых совпала цена, отсортировать по артикулу:

```
SELECT * FROM PRODUCTS ORDER BY 3 DESC, Product_ID
```

Product_ID	Product_Name	Price
4	Папиросы "Ира"	0.80
2	Табак резаный	0.50
5	Папиросы "Пушка"	0.40
1	Табак листовой	0.20
6	Тетрадка в клетку 12 л	0.20
3	Сигары "Шик"	
7	Ручка шариковая	

До сих пор мы выводили все строки таблицы, лишь меняя их порядок или выводя некоторые атрибуты. Теперь научимся ограничивать выводимые строки, указывая условия выборки. Простое условие поиска составляется из имен столбцов, констант и операций сравнения.

```
SELECT * FROM PRODUCTS WHERE Price<0.5
```

Product_ID	Product_Name	Price
1	Табак листовой	0.20
5	Папиросы "Пушка"	0.40
6	Тетрадка в клетку 12 л	0.20

Условий выборки может быть несколько. Простые условия могут быть связаны логическими операциями `AND, OR, NOT`:

```
SELECT * FROM PRODUCTS WHERE PRICE<0.5 AND PRICE>0.3
```

Product_ID	Product_Name	Price
5	Папиросы "Пушка"	0.40

```
SELECT * FROM PRODUCTS WHERE PRICE>=0.5 OR PRICE<0.3
```

Product_ID	Product_Name	Price
1	Табак листовой	0.20
2	Табак резаный	0.50
4	Папиросы "Ира"	0.80
6	Тетрадка в клетку 12 л	0.20

```
SELECT * FROM PRODUCTS WHERE PRICE>0.4 AND NOT PRODUCT_ID=2
```

Product_ID	Product_Name	Price
4	Папиросы "Ира"	0.80

Можно использовать сравнение с незаданными значениями, которые обозначаются ключевым словом `NULL`. Для них используется особый операцию сравнения `IS`:

```
SELECT * FROM PRODUCTS WHERE Price IS NULL
```

Product_ID	Product_Name	Price
3	Сигары "Шик"	
7	Ручка шариковая	

Или `IS NOT`:

```
SELECT * FROM PRODUCTS WHERE Price IS NOT NULL
```

Product_ID	Product_Name	Price
1	Табак листовой	0.20
2	Табак резаный	0.50
4	Папиросы "Ира"	0.80
5	Папиросы "Пушка"	0.40
6	Тетрадка в клетку 12 л	0.20

Если нужно сравнивать значение столбца с несколькими значениями на равенство, то можно использовать следующую конструкцию запроса:

```
SELECT * FROM PRODUCTS WHERE Product_id IN (1, 5, 6)
```

Product_ID	Product_Name	Price
1	Табак листовой	0.20
5	Папиросы "Пушка"	0.40
6	Тетрадка в клетку 12 л	0.20

Так мы найдем все продукты, где артикул принимает одно из указанных значений. Для поиска текстовых полей можно использовать шаблоны при помощи операции `LIKE`.

В шаблоне кроме искомых слов могут присутствовать специальные символы % — процент и _ — подчеркивание.

```
SELECT * FROM PRODUCTS WHERE Product_name LIKE 'Папиросы%'
```

Product_ID	Product_Name	Price
4	Папиросы "Ира"	0.80
5	Папиросы "Пушка"	0.40

На месте подчеркивания в шаблоне может быть любой символ; а знак процента заменяет любую (в том числе и пустую) последовательность символов.

Вспомним схему нашей базы данных. У нас есть таблица Покупатели, Товары и Заказы Таблица Заказы связывает таблицы Покупатель и Товар. Как же нам проследить эту связь?

Например, нам захочется узнать, какие заказы относятся к товару Карандаш. Конечно, можно решать задачу за несколько шагов. Сначала найдем код товара Карандаш

```
SELECT Product_ID FROM PRODUCTS WHERE Product_Name='Карандаш'
```

Затем найдем заказы из таблицы ORDERS, соответствующие найденному Коду товара.

```
SELECT * FROM ORDERS WHERE Product_ID=5
```

А можно ли найти нужную информацию за одну команду? Конечно.

```
SELECT * FROM ORDERS WHERE
Product_ID=(SELECT Product_ID FROM PRODUCTS WHERE
Product_Name='Карандаш')
```

Мы воспользовались вложенным запросом, когда один запрос содержит в себе другие. Вложенный запрос может быть использован вместо таблицы или множества после предложений FROM или WHERE, или в качестве выражения после предложения SELECT.

Например, при помощи вложенных запросов мы можем для каждого заказа узнать название товара.

```
SELECT Order_ID,
(SELECT Product_Name FROM PRODUCTS WHERE
PRODUCTS.Product_id=ORDERS.Product_id) FROM ORDERS
```

Вложенный запрос возвращает название товара, соответствующего коду товара. Заметим, что перед именем поля `Product_ID` появилось имя таблицы. На самом деле, можно было бы всегда указывать имя таблицы перед названием столбца. Но обязательно указывать это нужно делать, если в запросе участвуют две таблицы и у них есть столбцы с совпадающими именами. Без указания таблицы неясно, о каком столбце идет речь.

Попробуем переписать этот запрос иначе. Мы хотим соединить строки таблицы ORDERS и таблицы PRODUCTS. Критерий соединения — код товара.

Для этого есть специальный тип запроса — с соединением.

```
SELECT * FROM ORDERS JOIN PRODUCTS ON
PRODUCTS.Product_id=ORDERS.Product_id
```

Строки двух таблиц соединились по совпадающим значениям поля `Product_ID`. Вы заметили, что столбец `Product_ID` повторяется в выборке два раза? Давайте будем выводить его один раз.

```
SELECT PRODUCTS.*, Order_ID, Customer_ID, Quantity
FROM ORDERS JOIN PRODUCTS ON
PRODUCTS.Product_id=ORDERS.Product_id
```

А что будет с товарами, которые никто еще ни разу не заказал? Получается, такие строки таблицы PRODUCTS не удастся соединить, так как в таблице ORDERS нет ни одной строки с таким `Product_ID`. Поэтому такие товары не попадут в итоговую выборку. Поэтому соединение JOIN еще называют внутренним. Если нам нужно, чтобы в итоговую выборку при соединении попали все товары, то нужно использовать полное соединение — `FULL JOIN`:

```
SELECT * FROM ORDERS FULL JOIN PRODUCTS ON
PRODUCTS.Product_id=ORDERS.Product_id
```

Тогда товары, для которых нет ни одного заказа, будут дополнены незаданными значениями. Внутреннее соединение можно переписать иначе — через так называемое произведение таблиц. Что будет результатом запроса:

```
SELECT * FROM ORDERS, PRODUCTS
```

Оказывается, мы получим все парасочетания строк. Теперь можем выбрать из них те строки, в которых совпали поля `Product_ID`:

```
SELECT * FROM ORDERS, PRODUCTS WHERE
PRODUCTS.Product_id=ORDERS.Product_id
```

Часто нам требуется не просто вывести строки таблицы согласно заданных критериев, но и провести некоторую аналитику — определить количество строк в выборке, сумму или среднее значение по численному столбцу, определить минимум/максимум.

Для этого нам помогут встроенные функции агрегирования. Функцию агрегирования можно вызвать сразу после ключевого слова `SELECT`. Параметрами функций агрегирования могут быть столбцы или какие-то выражения, составленные из имен столбцов и констант).

Минимальный набор встроенных агрегатных функций в любой СУБД — это сумма, количество строк, среднее значение, минимум и максимум. Например, простейшая функция агрегирования считает количество строк.

```
SELECT COUNT(*) FROM ORDERS
```

Этот запрос определит общее количество строк таблицы `Orders`. Как найти цену самого дорогого товара? Воспользуемся функцией `MAX`.

```
SELECT MAX(Price) FROM PRODUCTS
```

Аналогично можно найти цену на самый дешевый товар, и среднюю цену товаров.

```
SELECT MIN(Price) FROM PRODUCTS
```

```
SELECT AVG(Price) FROM PRODUCTS
```

Можно добавить условие для выбора строк, к которым применяются функции агрегирования

```
SELECT AVG(Price) FROM PRODUCTS WHERE Price>0.4
```

Если мы используем функции агрегирования для всей таблицы, то после ключевого слова `SELECT` рядом с функциями агрегирования нельзя использовать обычные названия столбцов. Например, мы нашли цену самого дорогого товара

```
SELECT MAX(Price) FROM PRODUCTS
```

Но как найти сам товар? Хотелось бы написать запрос так:

```
SELECT MAX(Price), Product_ID, Product_Name FROM PRODUCTS
```

Но так писать нельзя. Почему такой запрос будет неправильным? Давайте представим, что у нас есть несколько товаров с максимальной ценой. Тогда непонятно, какой товар нужно вывести.

Напишем этот запрос правильно:

```
SELECT Product_ID, Product_Name FROM PRODUCTS
WHERE Price= (SELECT MAX(Price) FROM PRODUCTS)
```

Также можно группировать строки по определенному критерию, и для каждой группы вычислять агрегатные функции.

Например, сгруппируем строки таблицы Orders по товару — узнаем, сколько раз заказывали каждый товар.

```
SELECT COUNT(*) as total, Product_ID FROM Orders
GROUP BY Product_ID
```

Если мы используем группировку, то вместе с функциями агрегирования могут быть выбраны поля, по которым проводилась группировка. Давайте добавим в выборку название товара. Это можно сделать несколькими способами. Например, при помощи соединения:

```
SELECT COUNT(*) as total, Orders.Product_id, Product_Name
FROM Orders JOIN PRODUCTS on PRODUCTS.Product_id=ORDERS.Product_id
GROUP BY ORDERS.Product_id, Product_Name
```

Или с помощью произведения таблиц

```
SELECT COUNT(*) as total, Orders.Product_id, Product_Name
FROM Orders, PRODUCTS
WHERE PRODUCTS.Product_id=ORDERS.Product_id
GROUP BY ORDERS.Product_id, Product_Name
```

А можем использовать вложенный запрос

```
SELECT COUNT(*) as total, ORDERS.Product_id,
(SELECT Product_Name FROM PRODUCTS WHERE
PRODUCTS.Product_id=ORDERS.Product_id ) AS Name
FROM Orders
GROUP BY ORDERS.Product_id
```

Для групп можно указывать ограничения. Например, сосчитаем среднее количество единиц товара в каждом заказе, при этом будем рассматривать только те товары, на которые было больше одной покупки.

```
SELECT Product_id, AVG(Quantity) as Avg_Quantity
FROM Orders
GROUP BY ORDERS.Product_id
HAVING COUNT(*)>1
```

И напоследок еще один сложный запрос — как узнать стоимость каждого заказа? Нам нужно знать цену каждого товара — она есть в таблице PRODUCTS, и количество единиц товара в каждом заказе — эта информация хранится в таблице Orders.

```
SELECT Order_ID, Customer_ID, PRODUCTS.Product_Name,
Quantity*Price as Order_Price
FROM ORDERS JOIN PRODUCTS ON
PRODUCTS.Product_id=ORDERS.Product_id
WHERE Price IS NOT NULL
```

6 Объекты базы данных

Кроме таблиц, в которых непосредственно хранятся данные, СУБД хранит большое количество разных дополнительных объектов. Они нужны, чтобы кроме хранения данных предоставить пользователям множество полезных и удобных возможностей.

Начнем с представлений. При помощи представлений можно сохранить в базе данных сложный большой запрос, и потом обращаться к нему, просто используя его название.

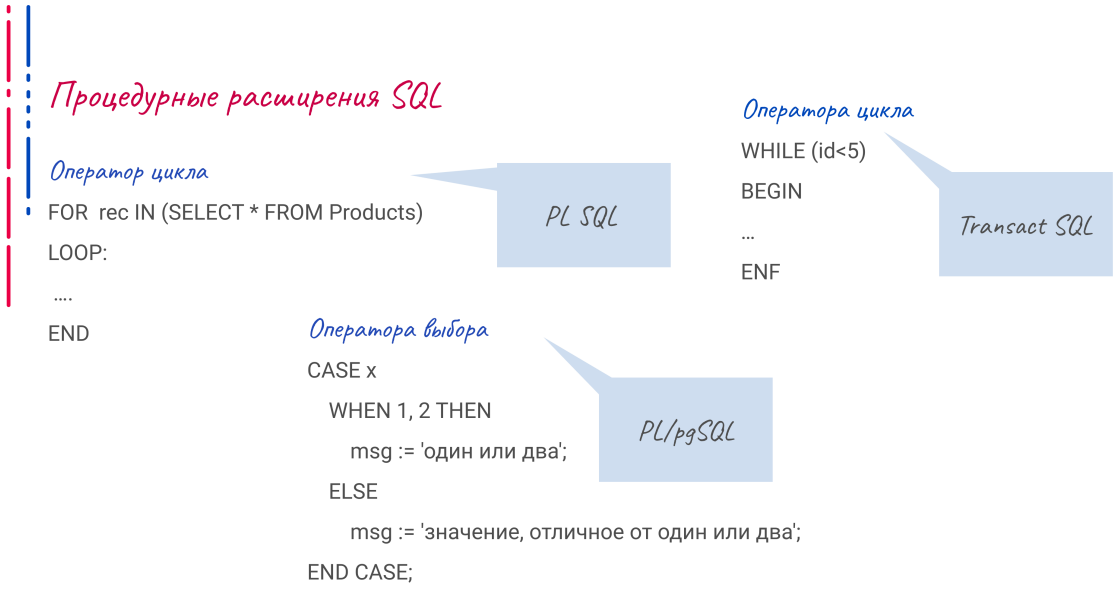
```
CREATE VIEW Products_ID_and_Name as
SELECT Product_ID, Product_Name FROM PRODUCTS

SELECT * FROM Products_ID_and_Name
```

Язык SQL является очень языком манипулирования данными, но для решения сложных задач обработки данных ему не хватает управляющих конструкций, например, условных операторов, операторов цикла и других., которые есть в других языках программирования. Поэтому многие СУБД имеют процедурные расширения языка SQL, которые представляют собой полноценный язык программирования, в котором можно использовать команды SQL. На таком языке можно писать процедуры и функции, постоянно хранящиеся в базе данных и исполняемые в среде СУБД.

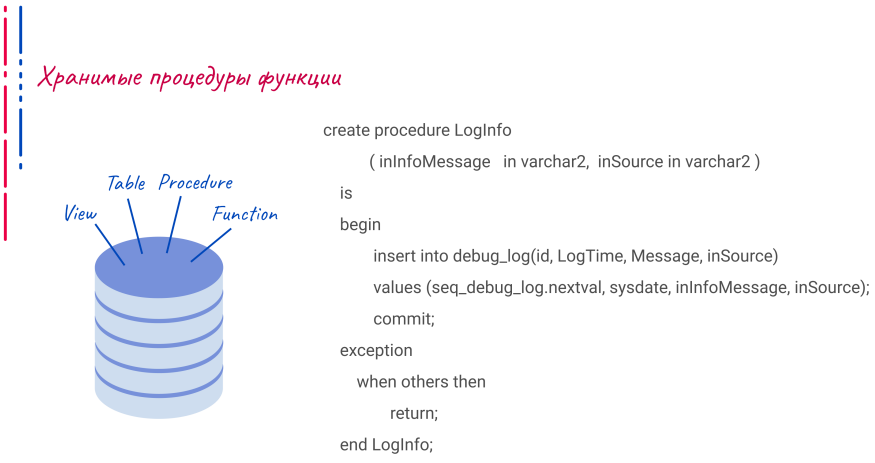
Процедуры и функции позволяют сохранить часто используемый набор операций с данными, которые могут быть встроены в различные приложения базы данных. Кроме того, СУБД содержат множество встроенных функций, например, математические, статистические, функции работы со строками, функции преобразования данных.

Правда каждая СУБД поддерживает свой собственный язык процедурного расширения SQL, что усложняет задачи переносимости программного обеспечения.



Как это работает? Давайте представим, что мы разрабатываем приложение для банка. И банк каждый день выдает множество кредитов. Понятно, что выдача кредита состоит из нескольких простых операций. Надо ввести информацию про клиента, проверить, что у него чистая кредитная история, определить его средний заработок и другие важные параметры, и в зависимости от этого определить максимальную сумму кредита. Все эти операции можно описать последовательно и сохранить в базе данных. Тогда в приложении можно будет просто вызывать функцию, задавая ее параметры.

Хранимые процедуры или функции могут быть вызваны из прикладной программы, которая решает задачи предметной области, используя базу данных. А есть особый вид процедур, которые не вызываются в явном виде, а исполняются автоматически в ответ на определенные изменения в базе данных. Такие процедуры называются триггерами.

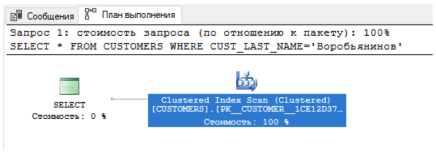


Триггеры — это особые процедуры, которые позволяют определить реакцию на события, происходящие в базе данных, и вызвать в ответ на это изменения в других данных. Например, в университет зачислили нового студента — добавили запись в таблицу Студент. Тогда надо выдать ему студенческий билет с уникальным номером. Надо проверить, иногородний ли он, и если да, то определить место в общежитии. Эту цепочку событий можно создать при помощи триггеров.

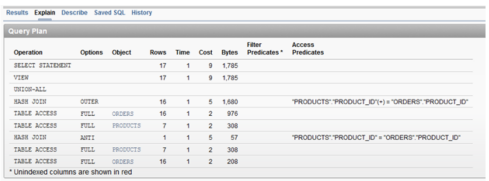
Очень важная часть СУБД — это оптимизатор выполнения запросов. Этот компонент определяет наиболее эффективный способ или план выполнения. Мы научились писать сложные запросы, которые могут собирать и агрегировать данные из нескольких таблиц. Эти запросы могут быть выполнены по-разному. Как, например, из Питера до Москвы можно добраться поездом, самолетом или автомобилем. Причем разница в производительности, или в скорости выполнения одного и того же запроса может отличаться на порядки. За счет чего это происходит? За счет изменения порядка элементар-

Оптимизатор запросов

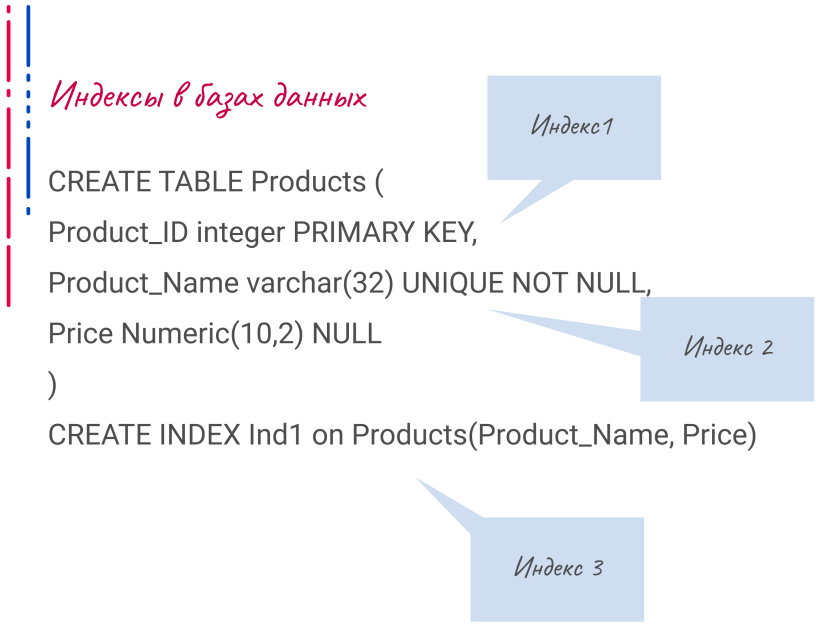
SELECT * FROM CUSTOMERS WHERE CUST_LAST_NAME='Воробьянинов'



SELECT * FROM ORDERS FULL JOIN PRODUCTS ON PRODUCTS.Product_id= ORDERS.Product_id



ных операций, на которые можно разбить запрос, и за счет использования дополнительных объектов, которые позволяют в большой таблице из миллиона строк быстро найти нужные значения по заданному критерию. Такие объекты называются индексами. Индексы создаются автоматически при описании таблицы для полей `PRIMARY KEY` и `UNIQUE`, а также их может создать администратор базы данных по другим столбцам, по которым часто производится поиск. Оптимизатор определяет, нужно ли производить прямое сканирование таблицы или использовать индексы на основании сведений о распределении данных в таблицах, к которым обращается команда. Еще один важный внутренний компонент СУБД, про который надо обязательно упомянуть — это



диспетчер транзакций. Вам наверняка термин «транзакция» известен из финансовой сферы — если при оплате товара вы получили сообщение о том, что транзакция прошла, значит деньги точно списаны с банковской карты. Сущность транзакции состоит в связывании нескольких команд в одну операцию по принципу все-или-ничего, т.е. транзакция выполняется либо целиком, либо не выполняется совсем. Диспетчер транзакций отслеживает все изменения, происходящие в БД, которые заносит в специальный журнал. В процессе работы с данными могут происходить разные неполадки — ошибки в программе, отключение электричества, повреждения оборудования. В случае любых сбоев и ошибок журнал транзакций позволяет восстановить корректное состояние базы данных. Также менеджер транзакций координирует параллельную работу с данными нескольких пользователей. Некоторые действия с данными нельзя производить одновременно — например, нельзя позволять нескольким пользователям бронировать одно и то же место в самолете на одном рейсе. В то же время не стоит запрещать одновременно просматривать данные. Менеджер транзакций при помощи установки блокировок и других механизмов предоставляет доступ к данным в разных режимах, чтобы действия пользователей не противоречили друг другу.

В целях безопасности и защиты данных не всем пользователям может быть открыт доступ ко всей информации, хранимой в базе данных. Например, не корректно, если персональные данные людей будут находиться в открытом доступе. А уж тем более нельзя давать всем возможность вносить корректировки в эти данные. Каждому пользователю может быть видна только часть хранимых данных, и доступны лишь некоторые операции. Права доступа к базе данных и набору данных, выдаются конкретным пользователям,

и определяют настройки прав доступа пользователя. Если есть много пользователей с одинаковым набором прав, то они объединяются в роли. Такие права доступа к данным обычно раздает администратор базы данных.

Таким образом мы показали, что реляционные СУБД являются мощным и надежным механизмом для хранения и манипулирования данными. Казалось бы, с их помощью все задачи с большими данными можно решить. Но оказывается, все не так просто. Реляционные СУБД хорошо подходят для хранения структурированных данных, а современные потоки данных порой не имеют четкой структуры — это документы, фотографии, видео- и аудиофайлы. Для их хранения и обработки разрабатываются специальные хранилища, которые обычно подходят для данных определенного вида. Называются такие системы NoSQL.