Лекция 1. Введение в структурированный язык запросов SQL

## Основные понятия

Всякая профессиональная *деятельность* так или иначе связана с информацией, с организацией ее сбора, хранения, выборки. Можно сказать, что неотъемлемой частью повседневной жизни стали *базы данных*, для поддержки которых требуется некоторый организационный метод, или механизм. Такой механизм называется системой управления *базами данных* ( *СУБД* ). Итак, введем основные понятия.

***База данных (БД)*** – совместно используемый набор логически связанных данных (и их описание), предназначенный для удовлетворения информационных потребностей организации.

***СУБД*** (система управления *базами данных* ) – программное обеспечение, с помощью которого пользователи могут определять, создавать и поддерживать *базу данных*, а также получать к ней контролируемый доступ.

Системы управления *базами данных* существуют уже много лет, многие из них обязаны своим происхождением системам с неструктурированными файлами на больших ЭВМ. Наряду с общепринятыми современными технологиями в области систем управления *базами данных* начинают появляться новые направления, что обусловлено требованиями растущего бизнеса, все увеличивающимися объемами корпоративных данных и, конечно же, влиянием технологий *Internet*.

### Реляционные базы данных

Управление основными потоками информации осуществляется с помощью так называемых систем управления *реляционными базами данных* , которые берут свое начало в традиционных системах управления *базами данных*. Именно объединение *реляционных баз данных* и *клиент-серверных технологий* позволяет современному предприятию успешно управлять собственными данными, оставаясь конкурентоспособным на рынке товаров и услуг.

*Реляционные БД* имеют мощный теоретический фундамент, основанный на математической теории отношений. Появление теории *реляционных баз данных* дало толчок к разработке ряда языков *запросов*, которые можно отнести к двум классам:

алгебраические языки, позволяющие выражать *запросы* средствами специализированных операторов, применяемых к отношениям;

языки исчисления предикатов, представляющие собой набор правил для записи выражения, определяющего новое отношение из заданной совокупности существующих отношений. Следовательно, исчисление предикатов есть метод определения того отношения, которое желательно получить как ответ на *запрос* из отношений, уже имеющихся в *базе данных*.

В реляционной модели объекты реального мира и взаимосвязи между ними представляются с помощью совокупности связанных между собой *таблиц* (отношений).

Даже в том случае, когда функции *СУБД* используются для выбора информации из одной или нескольких *таблиц* (т.е. выполняется ***запрос*** ), результат также представляется в табличном виде. Более того, можно выполнить *запрос* с применением результатов другого *запроса*.

Каждая ***таблица*** БД представляется как совокупность ***строк*** и *столбцов*, где *строки* (записи) соответствуют экземпляру объекта, конкретному событию или явлению, а ***столбцы*** (поля) – атрибутам (признакам, характеристикам, параметрам) объекта, события, явления.

В каждой *таблице* БД необходимо наличие ***первичного ключа*** – так именуют поле или набор полей, однозначно идентифицирующий каждый экземпляр объекта или запись. Значение *первичного ключа* в *таблице* БД должно быть уникальным, т.е. в *таблице* не допускается наличие двух и более записей с одинаковыми значениями *первичного ключа* . Он должен быть минимально достаточным, а значит, не содержать полей, удаление которых не отразится на его уникальности.

### Реляционные связи между таблицами баз данных

*Связи* между объектами реального мира могут находить свое отражение в структуре данных, а могут и подразумеваться, т.е. присутствовать на неформальном уровне.

Между двумя или более *таблицами базы данных* могут существовать *отношения подчиненности* , которые определяют, что для каждой записи главной *таблицы* (называемой еще родительской) возможно наличие одной или нескольких записей в подчиненной *таблице* (называемой еще дочерней).

Выделяют три разновидности *связи* между *таблицами базы данных*:

"один–ко–многим";

"один–к–одному";

"многие–ко–многим".

### Отношение "один–ко–многим"

Отношение "один–ко–многим" имеет место, когда одной записи родительской *таблицы* может соответствовать несколько записей дочерней. *Связь* "один–ко–многим" иногда называют *связью* "многие–к–одному". И в том, и в другом случае сущность *связи* между *таблицами* остается неизменной. *Связь* "один–ко–многим" является самой распространенной для *реляционных баз данных* . Она позволяет моделировать также иерархические структуры данных.

### Отношение "один–к–одному"

Отношение "один–к–одному" имеет место, когда одной записи в родительской *таблице* соответствует одна запись в дочерней. Это отношение встречается намного реже, чем отношение "один–ко–многим". Его используют, если не хотят, чтобы *таблица* БД "распухала" от второстепенной информации, однако для чтения связанной информации в нескольких *таблицах* приходится производить ряд операций чтения вместо одной, когда данные хранятся в одной *таблице*.

### Отношение "многие–ко–многим"

Отношение "многие–ко–многим" применяется в следующих случаях:

одной записи в родительской *таблице* соответствует более одной записи в дочерней;

одной записи в дочерней *таблице* соответствует более одной записи в родительской.

Всякую *связь* "многие–ко–многим" в *реляционной базе данных* необходимо заменить на *связь* "один–ко–многим" (одну или более) с помощью введения дополнительных *таблиц*.

## Стандарт и реализация языка SQL

Рост количества данных, необходимость их хранения и обработки привели к тому, что возникла потребность в создании *стандартного языка баз данных*, который мог бы функционировать в многочисленных компьютерных системах различных видов. Действительно, с его помощью пользователи могут манипулировать данными независимо от того, работают ли они на персональном компьютере, сетевой рабочей станции или универсальной ЭВМ.

Одним из языков, появившихся в результате разработки реляционной модели данных, является язык SQL (Structured Query Language), который в настоящее время получил очень широкое распространение и фактически превратился в *стандартный язык реляционных баз данных* . ***Стандарт*** на язык SQL был выпущен Американским национальным институтом стандартов (ANSI) в 1986 г., а в 1987 г. Международная организация стандартов (ISO) приняла его в качестве международного. Нынешний *стандарт* SQL известен под названием SQL/92.

С использованием любых стандартов связаны не только многочисленные и вполне очевидные преимущества, но и определенные недостатки. Прежде всего, стандарты направляют в определенное русло развитие соответствующей индустрии; в случае языка *SQL* наличие твердых основополагающих принципов приводит, в конечном счете, к совместимости его различных *реализаций* и способствует как повышению переносимости программного обеспечения и *баз данных* в целом, так и универсальности работы администраторов *баз данных*. С другой стороны, стандарты ограничивают гибкость и функциональные возможности конкретной *реализации*. Под ***реализацией языка*** SQL понимается программный продукт SQL соответствующего производителя. Для расширения функциональных возможностей многие разработчики, придерживающиеся принятых стандартов, добавляют к *стандартному языку SQL* различные расширения. Следует отметить, что стандарты требуют от любой законченной *реализации языка SQL* наличия определенных характеристик и в общих чертах отражают основные тенденции, которые не только приводят к совместимости между всеми конкурирующими *реализациями*, но и способствуют повышению значимости программистов *SQL* и пользователей *реляционных баз данных* на современном рынке программного обеспечения.

Все конкретные *реализации языка* несколько отличаются друг от друга. В интересах самих же производителей гарантировать, чтобы их *реализация* соответствовала современным стандартам *ANSI* в части переносимости и удобства работы пользователей. Тем не менее каждая *реализация SQL* содержит усовершенствования, отвечающие требованиям того или иного *сервера баз данных*. Эти усовершенствования или расширения языка *SQL* представляют собой дополнительные команды и опции, являющиеся добавлениями к стандартному пакету и доступные в данной конкретной *реализации*.

В настоящее время язык *SQL* поддерживается многими десятками *СУБД* различных типов, разработанных для самых разнообразных вычислительных платформ, начиная от персональных компьютеров и заканчивая мейнфреймами.

Все языки манипулирования данными, созданные для многих *СУБД* до появления *реляционных баз данных* , были ориентированы на *операции* с данными, представленными в виде логических записей файлов.

Разумеется, это требовало от пользователя детального знания организации хранения данных и серьезных усилий для указания того, какие данные необходимы, где они размещаются и как их получить.

Рассматриваемый язык *SQL* ориентирован на *операции* с данными, представленными в виде логически взаимосвязанных совокупностей *таблиц* -отношений. Важнейшая особенность его структур – ориентация на конечный результат обработки данных, а не на процедуру этой обработки. Язык *SQL* сам определяет, где находятся данные, индексы и даже какие наиболее эффективные последовательности операций следует использовать для получения результата, а потому указывать эти детали в *запросе* к *базе данных* не требуется.

## Введение в технологию клиент-сервер

В связи с расширением рынка информационных услуг производители программного обеспечения стали выпускать все более интеллектуальные, а значит, и объемные программные комплексы. Многие организации и отдельные пользователи часто не могли разместить приобретенные продукты на собственных ЭВМ. Для обмена информацией и ее распространения были созданы сети ЭВМ, а обобщающие программы и данные стали устанавливать на специальных файловых *серверах*.

Благодаря работающим с файловыми *серверами СУБД*, множество пользователей получают *доступ* к одним и тем же *базам данных*. Упрощается разработка различных автоматизированных систем управления организациями. Однако при таком подходе вся обработка *запросов* из программ или с терминалов пользовательских ЭВМ на них и выполняется, поэтому для реализации даже простого *запроса* необходимо считывать с файлового *сервера* или записывать на него целые файлы, а это ведет к конфликтным ситуациям и перегрузке сети. Для исключения указанных недостатков была предложена *технология клиент-сервер*, но при этом понадобился единый язык общения с *сервером* – выбор пал на *SQL*.

***Технология клиент-сервер*** означает такой способ взаимодействия программных компонентов, при котором они образуют единую систему. Как видно из самого названия, существует некий клиентский процесс, требующий определенных ресурсов, а также *серверный процесс*, который эти ресурсы предоставляет. Совсем необязательно, чтобы они находились на одном компьютере. Обычно принято размещать *сервер* на одном узле локальной сети, а *клиентов* – на других узлах.

В контексте *базы данных клиент* управляет пользовательским интерфейсом и логикой приложения, действуя как *рабочая станция*, на которой выполняются приложения *баз данных*. *Клиент* принимает от пользователя *запрос*, проверяет *синтаксис* и генерирует *запрос* к *базе данных* на языке *SQL* или другом языке *базы данных*, соответствующем логике приложения. Затем передает сообщение *серверу*, ожидает поступления ответа и форматирует полученные данные для представления их пользователю. *Сервер* принимает и обрабатывает *запросы* к *базе данных*, после чего отправляет полученные результаты обратно *клиенту*. Такая обработка включает проверку полномочий *клиента*, обеспечение требований целостности, а также выполнение *запроса* и обновление данных. Помимо этого поддерживается управление параллельностью и восстановлением.

*Архитектура клиент-сервер* обладает рядом преимуществ:

- обеспечивается более широкий доступ к существующим *базам данных*;
- повышается общая производительность системы: поскольку *клиенты* и *сервер* находятся на разных компьютерах, их процессоры способны выполнять приложения параллельно. Настройка производительности компьютера с *сервером* упрощается, если на нем выполняется только работа с *базой данных*;
- снижается стоимость аппаратного обеспечения; достаточно мощный компьютер с большим устройством хранения нужен только *серверу* – для хранения и управления *базой данных*;
- сокращаются коммуникационные расходы. Приложения выполняют часть операций на клиентских компьютерах и посылают через сеть только *запросы* к *базам данных*, что позволяет значительно сократить объем пересылаемых по сети данных;
- повышается уровень непротиворечивости данных. *Сервер* может самостоятельно управлять проверкой целостности данных, поскольку лишь на нем определяются и проверяются все ограничения. При этом каждому приложению не придется выполнять собственную проверку;
- архитектура *клиент-сервер* естественно отображается на архитектуру открытых систем.

Дальнейшее расширение двухуровневой архитектуры *клиент-сервер* предполагает разделение функциональной части прежнего, "толстого" (интеллектуального) *клиента* на две части. В трехуровневой архитектуре *клиент-сервер* "тонкий" (неинтеллектуальный) *клиент* на рабочей станции управляет только пользовательским интерфейсом, тогда как средний уровень обработки данных управляет всей остальной логикой приложения. Третий уровень – *сервер базы данных*. Эта трехуровневая *архитектура* оказалась более подходящей для некоторых сред – например, для сетей *Internet* и *intranet*, где в качестве *клиента* может выступать обычный Web-браузер.

## Типы команд SQL

Реализация в *SQL* концепции операций, ориентированных на табличное *представление данных* , позволила создать компактный язык с небольшим набором предложений. Язык *SQL* может использоваться как для выполнения *запросов* к данным, так и для построения прикладных программ.

Основные *категории команд* языка *SQL* предназначены для выполнения различных функций, включая построение объектов *базы данных* и манипулирование ими, начальную загрузку данных в *таблицы*, обновление и удаление существующей информации, выполнение *запросов* к *базе данных*, *управление доступом* к ней и ее общее *администрирование* .

Основные *категории команд* языка *SQL*:

DDL – язык определения данных;

DML – язык манипулирования данными;

DQL – язык *запросов* ;

*DCL* – язык управления данными;

команды администрирования данных;

команды управления транзакциями

### Определение структур базы данных (DDL)

Язык определения данных (Data Definition Language, DDL) позволяет создавать и изменять структуру объектов *базы данных*, например, создавать и удалять *таблицы*. Основными командами языка DDL являются следующие: `CREATE TABLE` , `ALTER TABLE` , `DROP TABLE` , `CREATE INDEX` , `ALTER INDEX` , `DROP INDEX` .

### Манипулирование данными (DML)

Язык манипулирования данными (*Data Manipulation Language* , DML) используется для манипулирования информацией внутри объектов *реляционной базы данных* посредством трех основных команд: `INSERT` , `UPDATE` , `DELETE` .

### Выборка данных (DQL)

Язык *запросов* DQL наиболее известен пользователям *реляционной базы данных* , несмотря на то, что он включает всего одну команду `SELECT` . Эта команда вместе со своими многочисленными опциями и предложениями используется для формирования *запросов* к *реляционной базе данных* .

### Язык управления данными (DCL - Data Control Language)

Команды управления данными позволяют управлять доступом к информации, находящейся внутри *базы данных*. Как правило, они используются для создания объектов, связанных с доступом к данным, а также служат для контроля над распределением привилегий между пользователями. Команды управления данными следующие: `GRANT` , `REVOKE` .

### Команды администрирования данных

С помощью команд администрирования данных пользователь осуществляет контроль за выполняемыми действиями и анализирует операции *базы данных* ; они также могут оказаться полезными при анализе производительности системы. Не следует путать администрирование данных с администрированием *базы данных*, которое представляет собой общее управление *базой данных* и подразумевает использование команд всех уровней.

### Команды управления транзакциями

Существуют следующие команды, позволяющие управлять транзакциями *базы данных*: `COMMIT` , `ROLLBACK` , `SAVEPOINT` , `SET TRANSACTION` .

## Преимущества языка SQL

Язык *SQL* является основой многих *СУБД*, т.к. отвечает за физическое структурирование и *запись* данных на *диск*, а также за *чтение данных* с диска, позволяет принимать *SQL- запросы* от других компонентов *СУБД* и пользовательских приложений. Таким образом, *SQL* – мощный инструмент, который обеспечивает пользователям, программам и вычислительным системам *доступ* к информации, содержащейся в *реляционных базах данных* .

Основные достоинства языка *SQL* заключаются в следующем:

стандартность – как уже было сказано, использование языка SQL в программах стандартизировано международными организациями;

независимость от конкретных *СУБД* – все распространенные *СУБД* используют SQL, т.к. *реляционную базу данных* можно перенести с одной *СУБД* на другую с минимальными доработками;

возможность переноса с одной вычислительной системы на другую – *СУБД* может быть ориентирована на различные вычислительные системы, однако приложения, созданные с помощью SQL, допускают

использование как для *локальных БД* , так и для крупных многопользовательских систем;

реляционная основа языка – SQL является языком *реляционных БД* , поэтому он стал популярным тогда, когда получила широкое распространение реляционная модель представления данных. Табличная структура *реляционной БД* хорошо понятна, а потому язык SQL прост для изучения;

возможность создания *интерактивных запросов* – SQL обеспечивает пользователям немедленный доступ к данным, при этом в интерактивном режиме можно получить результат *запроса* за очень короткое время без написания сложной программы;

возможность программного доступа к БД – язык SQL легко использовать в приложениях, которым необходимо обращаться к *базам данных* . Одни и те же операторы SQL употребляются как для интерактивного, так и программного доступа, поэтому части программ, содержащие обращение к БД, можно вначале проверить в интерактивном режиме, а затем встраивать в программу;

обеспечение различного представления данных – с помощью SQL можно представить такую структуру данных, что тот или иной пользователь будет видеть различные их представления. Кроме того, данные из разных частей БД могут быть скомбинированы и представлены в виде одной простой *таблицы*, а значит, представления пригодны для усиления защиты БД и ее настройки под конкретные требования отдельных пользователей;

возможность динамического изменения и расширения структуры БД – язык SQL позволяет манипулировать структурой БД, тем самым обеспечивая гибкость с точки зрения приспособленности БД к изменяющимся требованиям предметной области;

поддержка архитектуры *клиент-сервер* – SQL – одно из лучших средств для реализации приложений на платформе *клиент-сервер* . SQL служит связующим звеном между взаимодействующей с пользователем клиентской системой и серверной системой, управляющей БД, позволяя каждой из них сосредоточиться на выполнении своих функций.

Любой язык работы с *базами данных* должен предоставлять пользователю следующие возможности:

создавать *базы данных* и *таблицы* с полным описанием их структуры;

выполнять основные операции манипулирования данными, в частности, вставку, модификацию и удаление данных из *таблиц* ;

выполнять простые и сложные *запросы*, осуществляющие преобразование данных.

Кроме того, язык работы с *базами данных* должен решать все указанные выше задачи при минимальных усилиях со стороны пользователя, а структура и *синтаксис* его команд – достаточно просты и доступны для изучения. И наконец, он должен быть универсальным, т.е. отвечать некоторому признанному *стандарту*, что позволит использовать один и тот же *синтаксис* и структуру команд при переходе от одной *СУБД* к другой. Язык *SQL* удовлетворяет практически всем этим требованиям.

Язык *SQL* является примером языка с трансформирующейся ориентацией, или же языка, предназначенного для работы с *таблицами* с целью преобразования входных данных к требуемому выходному виду. Он включает только команды определения и манипулирования данными и не содержит каких-либо команд управления ходом вычислений. Подобные задачи должны решаться либо с помощью языков программирования или управления заданиями, либо интерактивно, в результате действий, выполняемых самим пользователем. *По* причине подобной незавершенности в плане организации вычислительного процесса язык *SQL* может использоваться двумя способами. Первый предусматривает интерактивную работу, заключающуюся во вводе пользователем с терминала отдельных *SQL*-операторов. Второй состоит во *внедрении SQL-операторов* в программы на *процедурных языках* . Язык *SQL* относительно прост в изучении. Поскольку это не *процедурный язык* , в нем необходимо указывать, какая *информация* должна быть получена, а не как ее можно получить. Иначе говоря, *SQL* не требует указания методов доступа к данным. Как и большинство современных языков, он поддерживает свободный формат записи операторов. Это означает, что при вводе отдельные элементы операторов не связаны с фиксированными позициями экрана. Язык *SQL* может использоваться широким кругом специалистов, включая администраторов *баз данных*, прикладных программистов и множество других конечных пользователей.

Язык *SQL* – первый и пока единственный *стандартный язык* для работы с *базами данных* , который получил достаточно широкое распространение. Практически все крупнейшие разработчики *СУБД* в настоящее время создают свои продукты с использованием языка *SQL* либо с *SQL*-интерфейсом. В него сделаны огромные инвестиции как со стороны разработчиков, так и со стороны пользователей. Он стал частью архитектуры приложений, является стратегическим выбором многих крупных и влиятельных организаций.

Язык *SQL* используется в других стандартах и даже оказывает влияние на разработку иных стандартов как инструмент определения (например, стандарт *Remote Data* Access, *RDA*). Создание языка способствовало не только выработке необходимых теоретических основ, но и подготовке успешно реализованных технических решений. Это особенно справедливо в отношении оптимизации *запросов*, методов распределения данных и реализации средств защиты. Начали появляться *специализированные реализации* языка, предназначенные для новых рынков: системы управления обработкой транзакций (OnLine *Transaction Processing*, *OLTP* ) и системы оперативной аналитической обработки или системы поддержки

*принятия решений* (OnLine Analytical *Processing*, *OLAP*). Уже известны планы дальнейших расширений *стандарта*, включающих поддержку распределенной обработки, объектно-ориентированного программирования, расширений пользователей и *мультимедиа*.

## Запись SQL-операторов

Для успешного изучения языка *SQL* необходимо привести краткое описание структуры *SQL*-операторов и нотации, которые используются для определения формата различных *конструкций языка*. Оператор *SQL* состоит из *зарезервированных слов*, а также из слов, определяемых пользователем. *Зарезервированные слова* являются постоянной частью языка *SQL* и имеют фиксированное *значение*. Их следует записывать в точности так, как это установлено, нельзя разбивать на части для переноса с одной *строки* на другую. Слова, определяемые пользователем, задаются им самим (в соответствии с синтаксическими правилами) и представляют собой *идентификаторы* или имена различных объектов *базы данных*. Слова в операторе размещаются также в соответствии с установленными синтаксическими правилами.

***Идентификаторы*** языка *SQL* предназначены для обозначения объектов в *базе данных* и являются именами *таблиц*, представлений, *столбцов* и других объектов *базы данных*. Символы, которые могут использоваться в создаваемых пользователем *идентификаторах* языка *SQL*, должны быть определены как набор символов. *Стандарт SQL* задает набор символов, который используется *по* умолчанию, – он включает строчные и прописные буквы латинского алфавита ( `A-Z` , `a-z` ), цифры ( `0-9` ) и символ подчеркивания ( `_` ). На формат *идентификатора* накладываются следующие ограничения:

*идентификатор* может иметь длину до 128 символов;

*идентификатор* должен начинаться с буквы;

*идентификатор* не может содержать пробелы.

```
<идентификатор>::=<буква>
        {<буква>|<цифра>}[,...n]
```

Большинство компонентов языка не чувствительны к регистру. Поскольку у языка *SQL* свободный формат, отдельные *SQL-операторы* и их последовательности будут иметь более читаемый вид при использовании отступов и выравнивания.

Язык, в терминах которого дается описание языка SQL, называется ***метаязыком*** . Синтаксические определения обычно задают с помощью специальной металингвистической символики, называемой ***Бэкуса-Науэра формами*** (БНФ). Прописные буквы используются для записи *зарезервированных слов* и должны указываться в операторах точно так, как это будет показано. Строчные буквы употребляются для записи слов, определяемых пользователем. Применяемые в нотации *БНФ* символы и их обозначения показаны в *таблице*.

Таблица 1.1.

| Символ | Обозначение |
|---|---|
| ::= | Равно по определению |
| \| | Необходимость выбора одного из нескольких приведенных значений |
| <...> | Описанная с помощью *метаязыка* структура языка |
| {...} | Обязательный выбор некоторой конструкции из списка |
| [...] | Необязательный выбор некоторой конструкции из списка |
| [,...n] | Необязательная возможность повторения конструкции от нуля до нескольких раз |

## Описание учебной базы данных

В дальнейшем изложении в качестве примера будет использоваться небольшая *база данных*, отражающая процесс поставки или продажи некоторого товара постоянным клиентам.

Исходя из анализа *предметной области*, можно выделить два типа сущностей – `ТОВАР` и `КЛИЕНТ` , которые *связаны* между собой отношением "многие–ко–многим", т.к. каждый *покупатель* может купить много наименований товара, а каждый *товар* может быть куплен многими покупателями. Однако реляционная *модель данных* требует заменить *отношение* "многие–ко-многим" на несколько отношений "один–ко-многим". Добавим еще один тип сущностей, отображающий процесс продажи товаров, – `СДЕЛКА` .

Установим *связи* между объектами. Один *покупатель* может неоднократно покупать товары, поэтому между объектами `КЛИЕНТ` и `СДЕЛКА` имеется *связь* "один–ко–многим". Каждое наименование товара может

неоднократно участвовать в сделках, в результате между объектами ТОВАР и СДЕЛКА имеется *связь* "*один-ко-многим*".

Определим атрибуты и свяжем их с сущностями и *связями*. К объекту ТОВАР относятся такие характеристики, как название, тип, цена, сорт. К объекту КЛИЕНТ – имя, отчество, фамилия, *фирма*, город, телефон. *Тип сущности* СДЕЛКА может быть охарактеризован такими признаками, как дата и количество проданного товара.

Важным этапом в создании *базы данных* является *определение* атрибутов, которые однозначно определяют каждый *экземпляр сущности*, т.е. выявление *первичных ключей*.

Для *таблицы* ТОВАР название не может служить *первичным ключом*, т.к. товары разных типов могут иметь одинаковые названия, поэтому введем *первичный ключ* КодТовара, под которым можно понимать, например, *артикул товара*. Точно так же ни Имя, ни Фирма, ни Город не могут служить *первичным ключом* в *таблице* КЛИЕНТ. Введем *первичный ключ* КодКлиента, под которым можно понимать номер паспорта, идентификационный номер налогоплательщика или любой другой *атрибут*, однозначно определяющий каждого клиента. Для *таблицы* СДЕЛКА *первичным ключом* является *поле* КодСделки, т.к. оно однозначно определяет дату, покупателя и другие элементы данных. В качестве *первичного ключа* можно было бы выбрать не одно *поле*, а некоторую совокупность полей, но для иллюстрации *конструкций языка* ограничимся простыми *первичными ключами*.

Установим *связи* между *таблицами*. Один *покупатель* может неоднократно покупать товары. Поэтому между *таблицами* КЛИЕНТ и СДЕЛКА имеется *связь* "один–ко–многим" *по* полю КодКлиента.

Каждый *покупатель* может приобрести несколько различных товаров. Поэтому между *таблицами* ТОВАР и СДЕЛКА имеется *связь* "один–ко–многим" *по* полю КодТовара.

Теперь нужно создать *связи* между *таблицами базы данных*. Для этого поместим копии *первичных ключей* из родительской *таблицы* (*таблицы* со стороны "один") в дочернюю *таблицу* (*таблицу* со стороны "много"). Для организации *связи* между *таблицами* ТОВАР и СДЕЛКА поместим копию поля КодТовара из *таблицы* ТОВАР в *таблицу* СДЕЛКА. Для организации *связи* между *таблицами* КЛИЕНТ и СДЕЛКА поместим копию поля КодКлиента из *таблицы* КЛИЕНТ в *таблицу* СДЕЛКА. Для *таблицы* СДЕЛКА поля КодКлиента и КодТовара являются внешними (чужими) ключами. В результате получим следующую структуру *базы данных*.

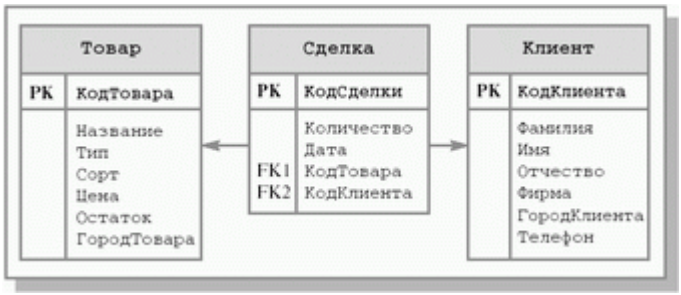


**Рис. 1.1.** Пример структуры базы данных.



## Задание: самостоятельная работа на тему: "История языка SQL"