# **Введение в базы данных.**

**Информационные системы** (ИС) – совокупность аппаратно-программных средств, задействованных для решения некоторой прикладной задачи. В организации, например, могут существовать информационные системы, на которых соответственно возложены следующие задачи: учет кадров и материально-технических средств, расчет с поставщиками и заказчиками, бухгалтерский учет и т. п.

**Банк данных –** является разновидностью ИС, в которой реализованы функции централизованного хранения и накопления обрабатываемой информации, организованной в одну или несколько баз данных.

Банк данных (БнД) в общем случае состоит из следующих компонентов:

* базы (нескольких баз) данных;
* системы управления базами данных (СУБД);
* словаря данных;
* администратора вычислительной системы;
* обслуживающего персонала.

**База данных** (БД) представляет собой совокупность специальным образом организованных данных, хранимых в памяти вычислительной системы и отображающих состояние объектов, и их взаимосвязей в рассматриваемой предметной области.

БД бывают централизованными (хранятся на одном компьютере) и распределенными (хранятся на нескольких компьютерах некоторой сети).

Логическую структуру хранимых в базе данных называют **моделью представления данных.** К основным моделям представления данных (моделям данных) относятся следующие:

* Иерархическая;
* Сетевая;
* Реляционная;
* Постреляционная;
* Многомерная;
* Объектно-ориентированная.

**Система управления базами данных (СУБД) — это комплекс языковых и программных средств, предназначенный для создания, ведения и совместного использования БД многими пользователями.** Обычно СУБД различают по используемой модели данных. Так, СУБД, основанные на использовании реляционной модели данных, называют реляционными СУБД.

Одними из первых СУБД являются следующие системы: IMS (IBM, 1968 г.), IDMS (Cullinet, 1971 г.), ADABAS (Software AG, 1969 г.) и ИНЭС (ВНИИСИ АН СССР, 1976 г.). Количество современных систем управления базами данных исчисляется тысячами.

**Приложение** представляет собой программу или комплекс программ, обеспечивающих автоматизацию обработки информации для прикладной задачи. Нами рассматриваются приложения, использующие БД. Приложения могут создаваться в среде или вне среды СУБД — с помощью системы программирования, использующей средства доступа к БД, к примеру, Delphi или С++ Builder. Приложения, разработанные в среде СУБД, часто называют приложениями СУБД, а приложения, разработанные вне СУБД, — внешними приложениями.

Для работы с базой данных зачастую достаточно средств СУБД и не нужно использовать приложения, создание которых обычно требует программирования. Приложения разрабатывают главным образом в случаях, когда требуется сделать работу пользователей более удобной или автоматизировать рутинные операции с БД.

**Словарь данных** (СД) представляет собой подсистему БнД, предназначенную для централизованного хранения информации о структурах данных, взаимосвязях файлов БД друг с другом, типах данных и форматах их представления, принадлежности данных пользователям, кодах защиты и разграничения доступа и т. п.

Словарь данных, иначе называемый системным каталогом, как следует из определения, является хранилищем служебной информации о данных в базе («данных о данных», или метаданных).

Функционально СД присутствует во всех БнД, но не всегда выполняющий эти функции компонент имеет именно такое название. Чаще всего функции СД выполняются СУБД и вызываются из основного меню системы или реализуются с помощью ее утилит.

Если СД является частью БД, то его называют интегрированным СД, в противном случае СД является автономным.

Автономные словари данных обычно используют не только в интересах собственно данных базы, но и в целях управления другими информационными ресурсами организаций при разработке структур баз данных на этапе проектирования, для ведения документации, управления проектами и т. д.

Стандартизация интерфейса СД привела к разработке службы словаря информационных ресурсов (Information Resource Dictionary System - IRDS). Служба IRDS имеет четыре интерфейса: графический, командный язык, экспорта/импорта и прикладных программ. Реализация IRDS представляет собой программный инструмент для унифицированного управления различными информационными ресурсами организации группами пользователей и приложениями. Введение IRDS может быть целесообразно на ранних этапах проектирования БД организации, когда необходимо отложить привязку БД к конкретной СУБД. Кроме того, с помощью служб IRDS можно переносить информацию между IRDS-совместимыми СД различных СУБД (независимо от используемой в них модели данных).

**Администратор базы данных** (АБД) есть лицо или группа лиц, отвечающих за выработку требований к БД, ее проектирование, создание, эффективное использование и сопровождение. В процессе эксплуатации АБД обычно следит за функционированием информационной системы, обеспечивает защиту от несанкционированного доступа, контролирует избыточность, непротиворечивость, сохранность и достоверность хранимой в БД информации. Для однопользовательских информационных систем функции АБД обычно возлагаются на лиц, непосредственно работающих с приложением БД.

В вычислительной сети АБД, как правило, взаимодействует с администратором сети. В обязанности последнего входят контроль за функционированием аппаратно-программных средств сети, реконфигурация сети, восстановление программного обеспечения после сбоев и отказов оборудования, профилактические мероприятия и обеспечение разграничения доступа.

**Вычислительная система** (ВС) представляет собой совокупность взаимосвязанных и согласованно действующих ЭВМ или процессоров и других устройств, обеспечивающих автоматизацию процессов приема, обработки и выдачи информации потребителям. Поскольку основными функциями БнД являются хранение и обработка данных, то используемая ВС, наряду с приемлемой мощностью центральных процессоров (ЦП) должна иметь достаточный объем оперативной и внешней памяти прямого доступа.

**Обслуживающий персонал** выполняет функции поддержания технических и программных средств в работоспособном состоянии. Он проводит профилактические, регламентные, восстановительные и другие работы по планам, а также по мере необходимости.

# **Классификация баз данных.**



# **Классификация систем управления базами данных.**

В качестве основных классификационных признаков можно использовать следующие:

* вид программы;
* характер использования;
* модель данных.

Названные признаки существенно влияют на целевой выбор СУБД и эффективность использования разрабатываемой информационной системы.

***Классификация СУБД.*** В общем случае под СУБД можно понимать любой программный продукт, поддерживающий процессы создания, ведения и использования БД. Рассмотрим, какие из имеющихся на рынке программ имеют отношение к БД и в какой мере они связаны с базами данных.

К СУБД относятся следующие основные виды программ:

•полнофункциональные СУБД;

•серверы БД;

•клиенты БД;

•средства разработки программ работы с БД.

***Полнофункциональные СУБД*** (ПФСУБД) представляют собой традиционные СУБД, которые сначала появились для больших машин, затем для мини-машин и для ПЭВМ. Из числа всех СУБД современные ПФСУБД являются наиболее многочисленными и мощными по своим возможностям. К ПФСУБД относятся, например, такие пакеты, как Clarion Database Developer, DataEase, DataFlex, dBase IV, Microsoft Access, Microsoft FoxPro и Paradox R:BASE.

Обычно ПФСУБД имеют развитый интерфейс, позволяющий с помощью команд меню выполнять основные действия с БД: создавать и модифицировать структуры таблиц, вводить данные, формировать запросы, разрабатывать отчеты, выводить их на печать и т. п. Для создания запросов и отчетов не обязательно программирование, а удобно пользоваться языком QBE (Query By Example — формулировки запросов по образцу). Многие ПФСУБД включают средства программирования для профессиональных разработчиков.

Некоторые системы имеют в качестве вспомогательных и дополнительные средства проектирования схем БД или CASE-подсистемы. Для обеспечения доступа к другим БД или к данным SQL-серверов полнофункциональные СУБД имеют факультативные модули.

***Серверы БД*** предназначены для организации центров обработки данных в сетях ЭВМ. Эта группа БД в настоящее время менее многочисленна, но их количество постепенно растет. Серверы БД реализуют функции управления базами данных, запрашиваемые другими (клиентскими) программами обычно с помощью операторов SQL.

Примерами серверов БД являются следующие программы: NetWare SQL (Novell), MS SQL Server (Microsoft), InterBase (Borland), SQLBase Server (Gupta), Intelligent Database (Ingress).

В роли ***клиентских программ*** для серверов БД в общем случае могут использоваться различные программы: ПФСУБД, электронные таблицы, текстовые процессоры, программы электронной почты и т. д. При этом элементы пары «клиент — сервер» могут принадлежать одному или разным производителям программного обеспечения.

В случае, когда клиентская и серверная части выполнены одной фирмой, естественно ожидать, что распределение функций между ними выполнено рационально. В остальных случаях обычно преследуется цель обеспечения доступа к данным «любой ценой». Примером такого соединения является случай, когда одна из полнофункциональных СУБД играет роль сервера, а вторая СУБД (другого производителя) — роль клиента. Так, для сервера БД SQL Server (Microsoft) в роли клиентских (фронтальных) программ могут выступать многие СУБД, такие как dBASE IV, Blyth Software, Paradox, DataEase, Focus, 1-2-3, MDBS III, Revelation и другие.

Средства разработки программ работы с БД могут использоваться для создания разновидностей следующих программ:

•клиентских программ;

•серверов БД и их отдельных компонентов;

•пользовательских приложений.

Программы первого и второго вида довольно малочисленны, так как предназначены, главным образом, для системных программистов. Пакетов третьего вида гораздо больше, но меньше, чем полнофункциональных СУБД.

К средствам разработки пользовательских приложений относятся системы программирования, например, Clipper, разнообразные библиотеки программ для различных языков программирования, а также пакеты автоматизации разработок (в том числе систем типа клиент-сервер). В числе наиболее распространенных можно назвать следующие инструментальные системы: Delphi и Power Builder (Borland), Visual Studio (Microsoft), SILVERRUN (Computer Advisers Inc.), S-Designor ( S D P и Powersoft) и ERwin (LogicWorks).

Если говорить о конкретных системах программирования (для языков С++, С#, Visual Basic, Java и др.), то все они содержат некоторые средства доступа к наиболее широко используемым БД.

Кроме перечисленных средств, для управления данными и организации обслуживания БД используются различные дополнительные средства, к примеру мониторы транзакций.

По характеру использования СУБД делят на *персональные* и *многопользовательские.*

***Персональные СУБД***обычно обеспечивают возможность создания персональных БД и недорогих приложений, работающих с ними. Персональные СУБД или разработанные с их помощью приложения зачастую могут выступать в роли клиентской части многопользовательской СУБД. К персональным СУБД, например, относятся Visual FoxPro, Paradox, Clipper, dBase, Access и др.

***Многопользовательские СУБД***включают в себя сервер БД и клиентскую часть и, как правило, могут работать в неоднородной вычислительной среде (с разными типами ЭВМ и операционными системами). К многопользовательским СУБД относятся, например, СУБД Oracle и Informix.

В зависимости от способа хранения и обработки БД (централизованного или децентрализованного) СУБД можно разделить на два класса: **централизованные** (или обычные) СУБД и **децентрализованные** (или распределенные) СУБД. В обычных СУБД данные хранятся в том же месте, где и программы их управления. В распределенных СУБД как программное обеспечение, так и данные распределены по узлам сети. Распределенные СУБД могут быть однородными или неоднородными. Неоднородность СУБД может проявляться в отличии поддерживаемых моделей данных, типов данных, языков запросов, фирм-разработчиков и т. д.

Одной из разновидностей, распределенных СУБД являются ***мультибазовые системы****,* в которых управление каждым из узлов осуществляется автономно. В мультибазовых СУБД производится такая интеграция локальных систем, при которой не требуется изменение существующих СУБД и в то же время конечным пользователям предоставляется доступ к совместно используемым данным. Пользователи локальных СУБД получают возможность управлять данными собственных узлов без централизованного контроля, который присутствует в обычных распределенных СУБД. Примером мультибазовой СУБД является система UniSQL компании Cincom Corporation.

В зависимости от возможности распараллеливания процесса обработки данных выделяют *СУБД с* ***последовательной и параллельной обработкой*** *(параллельные СУБД).*Параллельные СУБД функционируют в многопроцессорной вычислительной системе (как правило, со множеством устройств хранения данных) или в сети компьютеров.

По используемой модели данных СУБД (как и БД), разделяют на

иерархические,

сетевые,

реляционные,

объектно-ориентированные и другие типы.

Некоторые СУБД могут одновременно поддерживать несколько моделей данных.

# **Функции СУБД.**

С точки зрения пользователя, СУБД реализует ***функции хранения, изменения*** (пополнения, редактирования и удаления) и ***обработки информации***, а также разработки и получения различных выходных документов.

Для работы с хранящейся в базе данных информацией СУБД предоставляет программам и пользователям следующие два типа языков*:*

•язык описания данных — высокоуровневый непроцедурный язык декларативного типа, предназначенный для описания логической структуры данных;

•язык манипулирования данными — совокупность конструкций, обеспечивающих выполнение основных операций по работе с данными: ввод, модификацию и выборку данных по запросам.

Названные языки в различных СУБД могут иметь отличия. Наибольшее распространение получили два стандартизованных языка: ***QBE (Query By Example)*** — язык запросов по образцу и ***SQL (Structured Query Language)*** - структурированный язык запросов. QBE в основном обладает свойствами языка *манипулирования* данными, SQL сочетает в себе свойства языков обоих типов —*описания* и *манипулирования* данными.

Перечисленные выше функции СУБД, в свою очередь, используют следующие основные функции более низкого уровня, которые назовем ***низкоуровневыми:***

•управление данными во внешней памяти;

•управление буферами оперативной памяти;

•управление транзакциями;

•ведение журнала изменений в БД;

•обеспечение целостности и безопасности БД.

Дадим краткую характеристику необходимости и особенностям реализации перечисленных функций в современных СУБД.

Реализация **функции *управления данными во внешней памяти***в разных системах может различаться и на уровне управления ресурсами (используя файловые системы ОС или непосредственное управление устройствами ПЭВМ), и по логике самих алгоритмов управления данными. В основном методы и алгоритмы управления данными являются «внутренним делом» СУБД и прямого отношения к пользователю не имеют. Качество реализации этой функции наиболее сильно влияет на эффективность работы специфических ИС, например, с огромными БД, со сложными запросами, большим объемом обработки данных.

Необходимость буферизации данных и как следствие реализации **функции *управления буферами***оперативной памяти обусловлено тем, что объем оперативной памяти меньше объема внешней памяти.

*Буферы*представляют собой области оперативной памяти, предназначенные для ускорения обмена между внешней и оперативной памятью. В буферах временно хранятся фрагменты БД, данные из которых предполагается использовать при обращении к СУБД или планируется записать в базу после обработки.

***Механизм транзакций*** используется в СУБД для поддержания целостности данных в базе. *Транзакцией*называется некоторая неделимая последовательность операций надданными БД, которая отслеживается СУБД от начала и до завершения. Если по каким-либо причинам (сбои и отказы оборудования, ошибки в программном обеспечении, включая приложение) транзакция остается незавершенной, то она отменяется.

В зависимости от времени, требуемого для выполнения, выделяют *обычные и продолжительные транзакции.*Продолжительные транзакции могут охватывать часы, дни и даже месяцы. Такие транзакции могут возникать в процессе проектирования и разработки сложных систем крупным коллективом людей. Кроме того, помимо обычных *плоских транзакций,*используется модель *вложенных транзакций.*В последнем случае транзакция рассматривается как набор взаимосвязанных подзадач (субтранзакций), каждая из которых также может состоять из произвольного количества субтранзакций.

Говорят, что транзакции присущи три основных свойства:

•атомарность (выполняются все входящие в транзакцию операции или ни одна);

•серализуемость (отсутствует взаимное влияние выполняемых в одно и то же время транзакций);

•долговечность (даже крах системы не приводит к утрате результатов зафиксированной транзакции).

***Ведение журнала изменений***в БД (журнализация изменений) выполняется СУБД для обеспечения надежности хранения данных в базе при наличии аппаратных сбоев и отказов, а также ошибок в программном обеспечении.

Журнал СУБД — это особая БД или часть основной БД, непосредственно недоступная пользователю и используемая для записи информации обо всех изменениях базы данных. В различных СУБД в журнал могут заноситься записи, соответствующие изменениям в СУБД на разных уровнях: от минимальной внутренней операции модификации страницы внешней памяти до логической операции модификации БД (например, вставки записи, удаления столбца, изменения значения в поле) и даже транзакции.

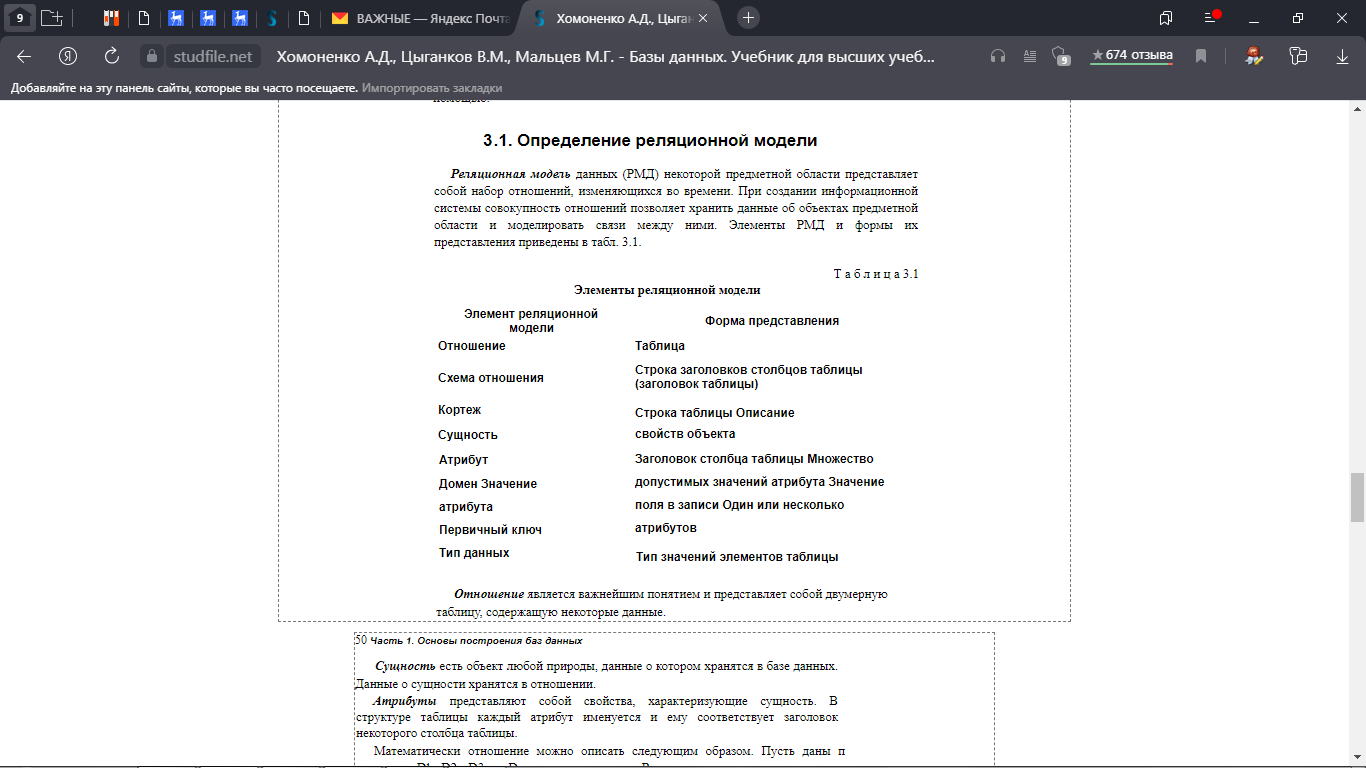
Для эффективной реализации функции ведения журнала изменений в БД необходимо обеспечить повышенную надежность хранения и поддержания в рабочем состоянии самого журнала. Иногда для этого в системе хранят несколько копий журнала.

***Обеспечение целостности***БД составляет необходимое условие успешного функционирования БД, особенно для случая использования БД в сетях. *Целостность*БД есть свойство базы данных, означающее, что в ней содержится полная, непротиворечивая и адекватно отражающая предметную область информация. Поддержание целостности БД включает проверку целостности и ее восстановление в случае обнаружения противоречий в базе данных. Целостное состояние БД описывается с помощью *ограничений целостности*в виде условий, которым должны удовлетворять хранимые в базе данные. Примером таких условий может служить ограничение диапазонов возможных значений атрибутов объектов, сведения о которых хранятся в БД, или отсутствие повторяющихся записей в таблицах реляционных БД.

***Обеспечение безопасности***достигается в СУБД шифрованием прикладных программ, данных, защиты паролем, поддержкой уровней доступа к базе данных и к отдельным ее элементам (таблицам, формам, отчетам и т.д.

# **Реляционная модель данных.**

***Реляционная модель***данных (РМД) некоторой предметной области представляет собой набор отношений, изменяющихся во времени. При создании информационной системы совокупность отношений позволяет хранить данные об объектах предметной области и моделировать связи между ними. Элементы РМД и формы их представления приведены в табл. 3.1.



***Отношение***представляет собой множество элементов, называемых кортежами. Наглядной формой представления отношения является привычная для человеческого восприятия двумерная таблица.

Таблица имеет строки (записи) и столбцы (колонки). Каждая строка таблицы имеет одинаковую структуру и состоит из полей. Строкам таблицы соответствуют кортежи, а столбцам — атрибуты отношения.

С помощью одной таблицы удобно описывать простейший вид связей между данными, а именно деление одного объекта (явления, сущности, системы и проч.), информация о котором хранится в таблице, на множество подобъектов, каждому из которых соответствует строка или запись таблицы. При этом каждый из подобъектов имеет одинаковую структуру или свойства, описываемые соответствующими значениями полей записей. Например, таблица может содержать сведения о группе обучаемых, о каждом из которых известны следующие характеристики: фамилия, имя и отчество, пол, возраст и образование. Поскольку в рамках одной таблицы не удается описать более сложные логические структуры данных из предметной области, применяют *связывание*таблиц.

Физическое размещение данных в реляционных базах на внешних носителях легко осуществляется с помощью обычных файлов.

*Достоинство*реляционной модели данных заключается в простоте, понятности и удобстве физической реализации на ЭВМ. Именно простота и понятность для пользователя явились основной причиной их широкого использования. Проблемы же эффективности обработки данных этого типа оказались технически вполне разрешимыми.

Основными *недостатками*реляционной модели являются следующие: отсутствие стандартных средств идентификации отдельных записей и сложность описания иерархических и сетевых связей.

Примерами зарубежных реляционных СУБД для ПЭВМ являются следующие: dBase!II Plus и dBase IY (фирма Ashton-Tate), DB2 (IBM), R:BASE (Microrim), FoxPro ранних версий и FoxBase (Fox Software), Paradox и dBASE for Windows (Borland), FoxPro более поздних версий, Visual FoxPro и Access (Microsoft), Clarion (Clarion Software), Ingres (ASK Computer Systems) и Oracle (Oracle).

К отечественным СУБД реляционного типа относятся системы: ПАЛЬМА (ИК АН УССР), а также система HyTech (МИФИ).

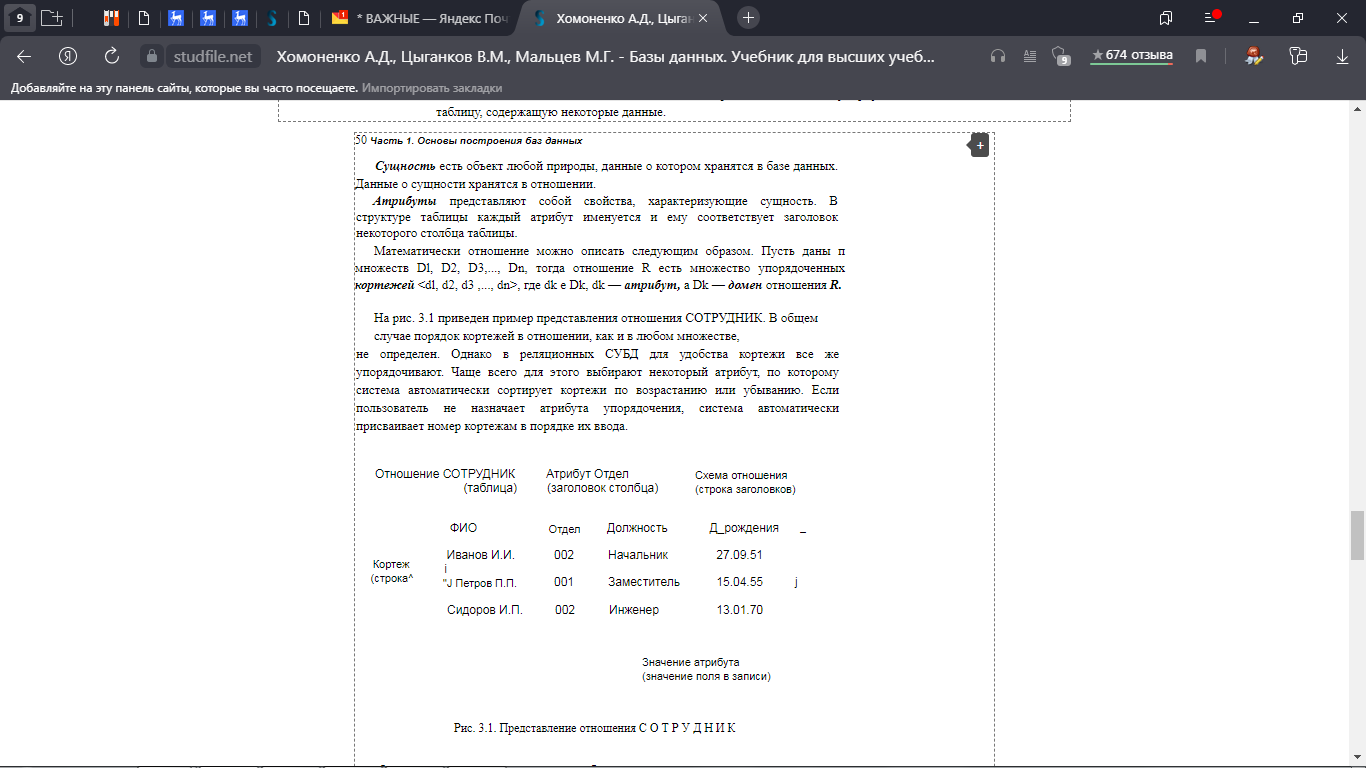
Заметим, что последние версии реляционных СУБД имеют некоторые свойства объектно-ориентированных систем. Такие СУБД часто называют объектно-реляционными. Примером такой системы можно считать продукты Oracle 8.x. Системы предыдущих версий вплоть до Oracle 7.x считаются «чисто» реляционными.

***Сущность***есть объект любой природы, данные о котором хранятся в базе данных. Данные о сущности хранятся в отношении.

***Атрибуты***представляют собой свойства, характеризующие сущность. В структуре таблицы каждый атрибут именуется и ему соответствует заголовок некоторого столбца таблицы.

Математически отношение можно описать следующим образом. Пусть даны *n* множеств D1, D2, D3,..., Dn, тогда отношение R есть множество упорядоченных ***кортежей***<d1, d2, d3 ,..., dn>, где dk — ***атрибут,*** a Dk — ***домен*** отношения ***R.***

На рис. 3.1 приведен пример представления отношения СОТРУДНИК. В общем случае порядок кортежей в отношении, как и в любом множестве, не определен. Однако в реляционных СУБД для удобства кортежи все же упорядочивают. Чаще всего для этого выбирают некоторый атрибут, по которому система автоматически сортирует кортежи по возрастанию или убыванию. Если пользователь не назначает атрибута упорядочения, система автоматически присваивает номер кортежам в порядке их ввода.



Формально, если переставить атрибуты в отношении, то получается новое отношение. Однако в реляционных БД перестановка атрибутов не приводит к образованию нового отношения.

***Домен***представляет собой множество всех возможных значений определенного атрибута отношения. Отношение СОТРУДНИК включает 4 домена. *Домен 1*содержит фамилии всех сотрудников, *домен 2 —*номера всех отделов фирмы, *домен 3 —*названия всех должностей, *домен 4 —*даты рождения всех сотрудников. Каждый домен образует значения одного типа данных, например, числовые или символьные.

Отношение СОТРУДНИК содержит 3 кортежа. Кортеж рассматриваемого отношения состоит из 4 элементов, каждый из которых выбирается из соответствующего домена. Каждому кортежу соответствует строка таблицы (рис. 3.1).

***Схема отношения {заголовок отношения)***представляет собой список имен атрибутов. Например, для приведенного примера схема отношения имеет вид С О Т Р У Д Н И К ( Ф И О , Отдел, Должность, Д\_Рождения). Множество собственно кортежей отношения часто называют содержимым {телом) отношения.

**Первичным ключом {ключом отношения, ключевым атрибутом)** называется атрибут отношения, однозначно идентифицирующий каждый из его кортежей. Например, в отношении СОТРУДНИК (ФИО, Отдел, Должность, Д\_Рождения) ключевым является атрибут «ФИО». Ключ может быть составным {сложным), то есть состоять из нескольких атрибутов.

Каждое отношение обязательно имеет комбинацию атрибутов, которая может служить ключом. Ее существование гарантируется тем, что отношение — это множество, которое не содержит одинаковых элементов — кортежей. То есть в отношении нет повторяющихся кортежей, а это значит, что по крайней мере вся совокупность атрибутов обладает свойством однозначной идентификации кортежей отношения. Во многих СУБД допускается создавать отношения, не определяя ключи.

Возможны случаи, когда отношение имеет несколько комбинаций атрибутов, каждая из которых однозначно определяет все кортежи отношения. Все эти комбинации атрибутов являются ***возможными ключами***отношения. Любой из возможных ключей может быть выбран как *первичный.*

Если выбранный первичный ключ состоит из минимально необходимого набора атрибутов, говорят, что он является ***не избыточным.***

Пусть в отношении R1 имеется *не ключевой*атрибут А, значения которого являются значениями *ключевого*атрибута В другого отношения R2. Тогда говорят, что атрибут А отношения R1 есть ***внешний ключ.***

*С*помощью внешних ключей устанавливаются связи между отношениями.

Поскольку не всякой таблице можно поставить в соответствие отношение, приведем условия, выполнение которых позволяет таблицу считать отношением.

1.Все строки таблицы должны быть уникальны, то есть не может быть строк с одинаковыми первичными ключами.

2.Имена столбцов таблицы должны быть различны, а значения их простыми, то есть недопустима группа значений в одном столбце одной строки.

3.Все строки одной таблицы должны иметь одну структуру, соответствующую именам и типам столбцов.

4.Порядок размещения строк в таблице может быть произвольным. Наиболее часто таблица с отношением размещается в отдельном файле.

В некоторых СУБД одна отдельная таблица (отношение) считается базой данных. В других СУБД база данных может содержать несколько таблиц.

# **Реляционная алгебра.**

**Реляционная алгебра** — замкнутая система операций над отношениями в реляционной модели данных. Операции реляционной алгебры также называют *реляционными операциями*.

**Операции реляционной алгебры**. Основные восемь операций реляционной алгебры были предложены Э.Коддом:

1. Объединение
2. Пересечение
3. Вычитание
4. Декартово произведение
5. Выборка
6. Проекция
7. Соединение
8. Деление

По справедливому замечанию Дейта, реляционная алгебра Кодда обладает несколькими недостатками. Во-первых, восемь перечисленных операций по охвату своих функций, с одной стороны, избыточны, так как минимально необходимый набор составляют пять операций: объединение, вычитание, произведение, проекция и выборка. Три другие операции (пересечение, соединение и деление) можно определить через пять минимально необходимых. Так, например, соединение — это проекция выборки произведения.

Во-вторых, этих восьми операций недостаточно для построения реальной СУБД на принципах реляционной алгебры. Требуются расширения, включающие операции: переименования атрибутов, образования новых вычисляемых атрибутов, вычисления итоговых функций, построения сложных алгебраических выражений, присвоения, сравнения и т. д.

Рассмотрим перечисленные операции более подробно, сначала — операции реляционной алгебры Кодда, а затем — дополнительные операции, введенные Дейтом.

*Операции реляционной алгебры Кодда можно разделить на две группы: базовые meopemuKO-множественные и специальные реляционные. Первая группа операций включает в себя классические операции теории множеств: объединение, разность, пересечение и произведение. Вторая группа представляет собой развитие обычных теоретико-множественных операций в направлении к реальным задачам манипулирования данными, в ее состав входят следующие операции:* ***проекция, селекция, деление и соединение****.*

Операции реляционной алгебры могут выполняться над одним отношением (например, проекция) или над двумя отношениями (например, объединение). В первом случае операция называется унарной, а во втором — бинарной. При выполнении бинарной операции участвующие в операциях отношения должны быть совместимы по структуре.

Совместимость структур отношений означает совместимость имен атрибутов и типов соответствующих доменов. Частным случаем совместимости является идентичность (совпадение). Для устранения конфликтов имен атрибутов в исходных отношениях (когда совпадение имен недопустимо), а также для построения произвольных имен атрибутов результирующего отношения применяется операция переименования атрибутов. Структура результирующего отношения по определенным правилам наследует свойства структур исходных отношений. В большинстве рассматриваемых бинарных реляционных операций будем считать, что заголовки исходных отношений идентичны, так как в этом случае не возникает проблем с заголовком результирующего отношения (в общем случае, заголовки могут не совпадать, тогда нужно оговаривать правила формирования заголовка отношения-результата).

**Проекция** является операцией, при которой из отношения выделяются атрибуты только из указанных доменов, то есть из таблицы выбираются только нужные столбцы, при этом, если получится несколько одинаковых кортежей, то в результирующем отношении остается только по одному экземпляру подобного кортежа.

**Выборка** — это операция, которая выделяет множество строк в таблице, удовлетворяющих заданным условиям. Условием может быть любое логическое выражение. В условии выборки мы можем использовать любое логическое выражение.

**Умножение или декартово произведение** является операцией, производимой над двумя отношениями, в результате которой мы получаем отношение со всеми доменами из двух начальных отношений. Кортежи в этих доменах будут представлять из себя все возможные сочетания кортежей из начальных отношений.

**Соединение и естественное соединение** – операция соединения обратна операции проекции и создает новое отношение из двух уже существующих. Новое отношение получается конкатенацией кортежей первого и второго отношений, при этом конкатенации подвергаются отношения, в которых совпадают значения заданных атрибутов.

**Пересечение и вычитание.** Результатом операции пересечения будет отношение, состоящее из кортежей, полностью входящих в состав обоих отношений. Результатом вычитания будет отношение, состоящее из кортежей, которые являются кортежами первого отношения и не являются кортежами второго отношения. Данные операции аналогичны таким же операциям над множествами.

***Дополнительные операции*** *реляционной алгебры, предложенные Дейтом,* включают следующие операции:

Операция ***переименования*** позволяет изменить имя атрибута отношения и имеет вид:

RENAME <исходное отношение> <старое имя атрибута> AS <новое имя атрибута>,

где <исходное отношение> задается именем отношения либо выражением реляционной алгебры. В последнем случае выражение заключают в круглые скобки.

Например, RENAME Город\_П AS Город\_размещения\_Поставщика.

***Замечание.***

Для удобства записи выражений одновременного переименования нескольких атрибутов Дейтом введена операция *множественного переименования,* которая представима следующим образом:

RENAME <отн.> <ст. имя атр.1> AS <нов. имя атр.1>,<ст. имя атр.2> AS <нов.  
имя агр.2>,... ,<ст. имя **aTp**.N> AS <нов. имя атр.№>.

Операция ***расширения*** порождает новое отношение, похожее на исходное, по отличающееся наличием добавленного атрибута, значения которого получаются путем некоторых скалярных вычислений. Операция расширения имеет вид:

EXTEND <исходное отношение> ADD <выражение> AS <новый атрибут>,

где к исходному отношению добавляется (ключевое слово ADD) <новый атрибут>, подсчитываемый по правилам, заданным <выражением>. Исходное отношение может быть задано именем отношения и с помощью выражения реляционной алгебры, заключенного в круглые скобки. При этом имя нового атрибута не должно входить в заголовок исходного отношения и не может использоваться в <выражении>. Помимо обычных арифметических операций и операций сравнения, в выражении можно использовать различные функции, называемые итоговыми, такие как: **COUNT (количество), SUM (сумма), AVG (среднее), МАХ** **(максимальное), MIN (минимальное).**

Например,

EXTEND (РJOIN SP) ADD (Вес \* Количество) AS 06щий\_Вес.

***Замечания.***

• Пользуясь операцией расширения, можно выполнить переименование атрибута. Для этого нужно в выражении указать имя атрибута, в конструкции AS определить новое имя этого атрибута, затем выполнить проекцию полученного отношения на множество атрибутов, исключая старый атрибут. Таким образом, запись (EXTEND S ADD Город\_П AS Город) [П#, Имя, Статус, Город] эквивалента конструкции S RENAME Город\_Г1 AS Город.

• Подобно тому, как это сделано для операции *переименования,*Дейт определил операцию *множественного расширения,*которая позволяет в одной синтаксической конструкции вычислять несколько новых атрибутов. Формально операция представима следующим образом:

EXTEND<отн> ADD<выр.1> AS<атр.1>,<выр.2>

AS <атр.2>,... **,<Bbip**.N> AS **<aTp**.N>.

Операция ***подведения итогов***SUMMARIZE выполняет «вертикальные» или групповые вычисления и имеет следующий формат:

SUMMARIZE <исх. отн.> BY (<список атрибутов>) ADD <выр.> AS <новый атрибут>,

где исходное отношение задается именем отношения либо заключенным в круглые скобки выражением реляционной алгебры, <список атрибутов> представляет собой разделенные запятыми имена атрибутов исходного отношения Al, А2,..., AN, <выр.> — скалярное выражение, аналогичное выражению операции EXTEND, а <новый атрибут> — имя формируемого атрибута. В списке атрибутов и в выражении не должен использоваться <новый атрибут>.

Результатом операции SUMMARIZE является отношение R с заголовком, состоящим из атрибутов списка, расширенного новым атрибутом. Для получения тела отношения R сначала выполняется проецирование (назовем проекцию R1) исходного отношения на атрибуты Al, А2,..., AN, после чего каждый кортеж проекции расширяется новым (N+1)-M атрибутом. Поскольку проецирование, как правило, приводит к сокращению количества кортежей по отношению к исходному отношению (удаляются одинаковые кортежи), то можно считать, что происходит своеобразное группирование кортежей исходного отношения: одному кортежу отношения R1 соответствует один или более (если было дублирование при проецировании) кортежей исходного отношения. Значение (N+l)-ro атрибута каждого кортежа отношения R формируется путем вычисления выражения над соответствующей этому кортежу группой кортежей исходного отношения.

Операция ***множественного подведения итогов****,* подобно соответствующим операциям переименования и расширения, выполняет одновременно несколько «вертикальных» вычислений и записывает результаты в отдельные новые атрибуты. Простейшим примером такой операции может служить следующая запись:

SUMMARIZE SP BY (Д#) ADD SUM Количество AS Общее\_число\_поставок AVG Количество AS Среднее\_число\_поставок.

К основным операторам, позволяющим изменять тело существующего отношения, отнесем операции реляционного *присвоения, вставки, обновления и удаления.*

Операцию ***присвоения***можно представить следующим образом:

<выражение-цель> := <выражение-источник>,

где оба выражения задают совместимые (точнее, эквивалентные) по структуре отношения. Типичный случай выражений: в левой части — имя отношения, а в правой — некоторое выражение реляционной алгебры. Выполнение операции присвоения сводится к замене предыдущего значения отношения на новое (начальное значение, если тело отношения было пустым), определенное выражением-источником.

С помощью операции присвоения можно не только полностью заменить все значения отношения-цели, но и добавить или удалить кортежи.

Операция ***вставки***INSERT имеет следующий вид:

INSERT <выражение-источник> INTO <выражение-цель>,

где оба выражения должны быть совместимы по структуре. Выполнение операции сводится к вычислению <выражение-источник> и вставке полученных кортежей в отношение, заданное <выражение-цель>.

Операция ***обновления*** UPDATE имеет следующий вид:

UPDATE <выражение-цель> <список элементов>,

где <список элементов> представляет собой последовательность разделенных запятыми операций присвоения <атрибут> := <скалярное выражение>. Результатом выполнения операции обновления является отношение, полученное после присвоения соответствующих значений атрибутам отношения, заданного целевым выражением.

Операция ***удаления***DELETE имеет следующий вид:

DELETE <выражение-цель>,

где <выражение-цель> представляет собой реляционное выражение, описывающее удаляемые кортежи.

Операция ***реляционного сравнения***может использоваться для прямого сравнения двух отношений. Она имеет синтаксис:

<выражение1> О <выражение2>,

где оба выражения задают совместимые по структуре отношения, а знак О — один из следующих операторов сравнения: = (равно), Ф (не равно), < (собственное подмножество), < (подмножество), > (надмножество), > (собственное надмножество).

***Основные правила записи выражений***. Как отмечалось, результатом произвольной реляционной операции является отношение, которое, в свою очередь, может участвовать в другой реляционной операции. Это свойство реляционной алгебры называется **свойством замкнутости**.

Свойство замкнутости позволяет записывать *вложенные выражения*реляционной алгебры, основой которых выступают рассмотренные ранее элементарные операции: объединение, проекция, пересечение, выборка и т. д.

При записи произвольного выражения реляционной алгебры надо принимать во внимание следующее.

1. В реляционной алгебре должен быть определен приоритет выполнения операций (например, операция пересечение более приоритетна чем операция объединение), который нужно учитывать при записи выражений. Для изменения порядка выполнения операций в выражениях можно использовать круглые скобки.

2. Существуют тождественные преобразования, позволяющие по-разному записывать одно и то же выражение. Например, следующие выражения эквивалентны (здесь А — отношение, С, CI, С2 — выражения):

3.Составляя выражение, нужно обеспечивать совместимость участвующих в операциях отношений. При необходимости изменения заголовков следует выполнять переименование атрибутов.

# **Модель клиент-сервер.**

Вычислительная модель **клиент—****сервер** исходно связана с парадигмой открытых систем, которая появилась в 90-х годах и быстро эволюционировала. Сам термин "клиент-сервер" исходно применялся к архитектуре программного обеспечения, которое описывало распределение процесса выполнения по принципу взаимодействия двух программных процессов, один из которых в этой модели назывался "клиентом", а другой — "сервером". Клиентский процесс запрашивал некоторые услуги, а серверный процесс обеспечивал их выполнение. При этом предполагалось, что один серверный процесс может обслужить множество клиентских процессов.

Ранее приложение (пользовательская программа) не разделялась на части, оно выполнялось некоторым монолитным блоком. Но возникла идея более рационального использования ресурсов сети. Действительно, при монолитном исполнении используются ресурсы только одного компьютера, а остальные компьютеры в сети рассматриваются как терминалы. Но теперь, в отличие от эпохи main-фреймов, все компьютеры в сети обладают собственными ресурсами, и разумно так распределить нагрузку на них, чтобы максимальным образом использовать их ресурсы.

**Системная архитектура «клиент-сервер».** Основные принципы системной архитектуры "клиент-сервер" заключаются в следующем: ИС разбивается на две части, которые могут выполняться в разных узлах сети - клиентскую и серверную части.

**Серверная сторона** функционирует на специализированном аппаратном комплексе, включающем в себя мощные аппаратные средства, требуемый набор стандартного программного обеспечения, систему управления базами данных и БД.

Прикладная программа или конечный пользователь взаимодействуют с **клиентской** частью системы, которая в простейшем случае обеспечивает просто надсетевой интерфейс. Клиентская сторона приложения функционирует на рабочем месте пользователя, в роли которого в подавляющем числе случаев выступает персональный компьютер.

Клиентская часть системы при необходимости обращается по сети (локальной или глобальной) к серверной части. При этом с точки зрения клиента и сервера взаимодействие осуществляется прозрачно. Сетевой компонент, реализующий это взаимодействие включает в себя совокупность необходимого сетевого оборудования, набор программных технологий, обеспечивающих передачу данных между узлами сети, а также собственно интерфейсный программный слой (протокол или протоколы) для обмена запросами и результатами их выполнения. **В качестве основного интерфейса между клиентской и серверной частями выступает язык баз данных SQL**.

Наиболее простая форма архитектуры «клиент-сервер» - это разделение вычислительной нагрузки между двумя отдельными процессами: клиентом и сервером. Причем между двумя этими процессами распределяются пять групп функций стандартного интерактивного приложения:

1) **функции диалога (Presentation Logic, PL) или компонент визуализации**. Компонент визуализации прикладной задачи осуществляет ввод информации пользователем с помощью тех или иных средств, а также вывод информации на экран и печать. Компонент визуализации для архитектуры клиент-сервер всегда исполняется на рабочем месте пользователя (поскольку должен же он наблюдать какие-либо результаты работы программы). Для организации презентационной логики преимущественно используется модель графического интерфейса пользователя GUI (User Interface Graphical) или Web-интерфейс;

2) **прикладные функции или это часть кода приложения с алгоритмами обработки данных (Business Logic, BL)**. Компонент прикладной логики решает собственно ту или иную задачу, связанную с обработкой данных в той или иной предметной области и представляет собой алгоритмы реакции приложения на действия пользователя или на внутренние события, правила обработки данных. Этот компонент может быть распределен между клиентской и серверной частью различным образом в зависимости от применяемой модели. Обычно этот код программируется на языках программирования высокого уровня: C, C++, Visual Basic, Object Pascal и т. п.;

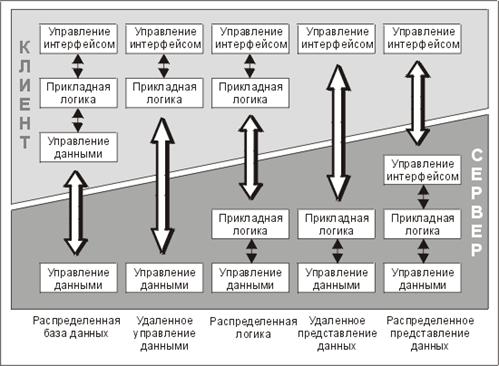
3) **функции обработки данных внутри приложения (Database Logic, DL)** - это часть кода приложения, связанная с выборкой и манипулированием данными (данными управляет собственно СУБД). Подъязыком является язык SQL.

4) **функции управления информационными ресурсами (Database Manager System, СУБД)**. Компонент хранения базы данных осуществляет физические операции, связанные с хранением данных, чтением информации из БД и записью в нее, связанных с решаемой приложением прикладной задачей. В архитектуре клиент-сервер этот компонент всегда выполняется на сервере;

5) **служебные функции**, играющие роль связок между предыдущими группами функций.

**Модели рабочих нагрузок в архитектуре клиент/сервер**. «Клиент-серверные» системы могут опираться на несколько типов распределения обязанностей между клиентом и сервером:

* «интеллектуальные» клиенты (толстый клиент - тонкий сервер);
* «интеллектуальный» сервер (тонкий клиент - толстый сервер);
* смешанные системы;
* многоуровневые системы.



Схему реализации выбирают на основе анализа требований:

* к сетевому трафику;
* к ресурсам клиента и сервера;
* к производительности базы данных.

«Интеллектуальные» клиенты - один из самых распространенных методов реализации «клиент-серверных» приложений. «Интеллектуальному» клиенту можно доверить выполнение сервисов представления, бизнес-логики и управление данными. В этом случае функции сервера ограничены поддержкой собственно базы данных (хранение данных). Вся информация обрабатывается локально на клиенте, что освобождает ресурсы сервера.

Достоинства «интеллектуальных» клиентов:

* простота архитектуры, что облегчает разработку и сопровождение системы;
* наличие хорошо известных и достаточно мощных средств разработки клиентских приложений;

Недостатки «интеллектуальных» клиентов:

* выполнение бизнес-правил на клиенте иногда увеличивает сетевой трафик из-за необходимости передавать клиенту все данные для принятия решения на основе правил;
* для модификации бизнес-логики необходимо повторное развертывание всех клиентов.

«Интеллектуальные» серверы. Перенеся все бизнес-правила на SQL Server, где они реализуются в виде хранимых процедур и триггеров БД, создается «интеллектуальный» сервер», сервер также отвечает за хранение и функционирование БД. Интеллект сервера проявляется, также, в способности манипулировать данными (выполнять SQL-запросы) и возвращать клиенту результирующий набор данных. На компьютере-клиенте реализуется только логика представления данных.

Достоинства:

* + увеличение производительности работы ИС: бизнес-логика выполняется в том же адресном пространстве, что и код доступа к базе данных, и, кроме того, тесно интегрирован с механизмом поиска данных. Это означает, что данные не нужно перемещать или копировать перед обработкой, а значит, сетевой трафик минимизируется;
  + на сервере легче обеспечить целостность данных;
  + при необходимости бизнес-логика модифицируется централизованно, без изменения клиентов.

Недостатки:

* + повышение требований к ресурсам сервера, где выполняются все запросы и манипуляции с данными.

Смешанные системы. Возможны и смешанные варианты, обладающие достоинствами, как интеллектуальных серверов, так и интеллектуальных клиентов. На сервере обычно разворачивают СУБД, логику манипулирования данными, и ту часть бизнес-логики, которая является общей для многих клиентов. На клиенте разворачивают логику представления данных и часть бизнес-логики, которая характерна именно для этого клиента.

Достоинства смешанных систем:

* + серверный код одновременно доступен многим клиентам, что снижает накладные расходы при выполнении однотипных запросов;
  + эффективность работы клиентов меньше зависит от сетевого трафика.

Недостатки:

* + бизнес-логика распределена между клиентом и сервером и при модернизации приложения требуется распространение новых версий клиентской части среди широкой аудитории.

Многоуровневая система (ее иногда называют трехуровневой) позволяет разделить пользовательский интерфейс, бизнес-правила и базу данных. В многоуровневой системе бизнес-правила помещают на сервере приложений – программный компонент, который размещают между сервером БД и клиентами. Клиент отвечает только за интерфейс с пользователем, сервер БД – за поддержание и функционирование самой БД. Промежуточный слой является для пользователя сервером, а для системы управления базами данных - клиентом. Клиенты при необходимости обращаются к серверу приложений, а тот в свою очередь обращается к серверу БД за данными, необходимыми для реализации запросов клиентов.

Достоинства многоуровневых систем:

* + разделение компонентов интерфейса, бизнес-правил и хранения данных, что повышает производительность работы сервера БД и его защиту от несанкционированного доступа;
  + централизованная модификация бизнес-правил;

Недостатки:

* + увеличивается сетевой трафик.
  + подобные системы намного сложнее в разработке, внедрении и эксплуатации и требуют значительных затрат и высококвалифицированного персонала.

# **Нормализация отношений.**

**Атрибут** — свойство некоторой сущности. Часто называется полем таблицы. **Домен атрибута** — множество допустимых значений, которые может принимать атрибут. **Кортеж** — конечное множество взаимосвязанных допустимых значений атрибутов, которые вместе описывают некоторую сущность (строка таблицы). **Отношение** — конечное множество кортежей (таблица). **Схема отношения** — конечное множество атрибутов, определяющих некоторую сущность. Иными словами, это структура таблицы, состоящей из конкретного набора полей. **Проекция** — отношение, полученное из заданного путём удаления и (или) перестановки некоторых атрибутов. **Функциональная зависимость** между атрибутами (множествами атрибутов) X и Y означает, что для любого допустимого набора кортежей в данном отношении: если два кортежа совпадают по значению X, то они совпадают по значению Y. **Нормальная форма** — требование, предъявляемое к структуре таблиц в теории реляционных баз данных для устранения из базы избыточных функциональных зависимостей между атрибутами (полями таблиц). **Метод нормальных форм (НФ)** состоит в сборе информации о объектах решения задачи в рамках одного отношения и последующей декомпозиции этого отношения на несколько взаимосвязанных отношений на основе процедур нормализации отношений. **Цель нормализации**: исключить избыточное дублирование данных, которое является причиной аномалий, возникших при добавлении, редактировании и удалении кортежей (строк таблицы). **Аномалией** называется такая ситуация в таблице БД, которая приводит к противоречию в БД либо существенно усложняет обработку БД. Причиной является излишнее дублирование данных в таблице, которое вызывается наличием функциональных зависимостей от не ключевых атрибутов. **Аномалии-модификации** проявляются в том, что изменение одних данных может повлечь просмотр всей таблицы и соответствующее изменение некоторых записей таблицы. **Аномалии-удаления** — при удалении какого-либо кортежа из таблицы может пропасть информация, которая не связана на прямую с удаляемой записью. **Аномалии-добавления** возникают, когда информацию в таблицу нельзя поместить, пока она не полная, либо вставка записи требует дополнительного просмотра таблицы. **Нормализация** — это процесс организации данных в базе данных, включающий создание таблиц и установление отношений между ними в соответствии с правилами, которые обеспечивают защиту данных и делают базу данных более гибкой, устраняя избыточность и несогласованные зависимости.

Избыточность данных приводит к непродуктивному расходованию свободного места на диске и затрудняет обслуживание баз данных. Например, если данные, хранящиеся в нескольких местах, потребуется изменить, в них придется внести одни и те же изменения во всех этих местах. Изменение адреса клиента гораздо легче реализовать, если в базе данных эти сведения хранятся только в таблице Customers и нигде больше.

Что такое «несогласованные зависимости»? Пользователь, которому нужно узнать, например, адрес определенного клиента, вполне обоснованно будет искать его в таблице Customers (клиенты), но искать в ней сведения о зарплате сотрудника, который работает с этим клиентом, не имеет смысла. Зарплата сотрудника связана с сотрудником (зависит от него), поэтому эти сведения следует хранить в таблице Employees (сотрудники). Несогласованные зависимости могут затруднять доступ к данным, так как путь к данным при этом может отсутствовать или быть неправильным.

Существует несколько правил нормализации баз данных. Каждое правило называется «нормальной формой». Если выполняется первое правило, говорят, что база данных представлена в «первой нормальной форме». Если выполняются три первых правила, считается, что база данных представлена в «третьей нормальной форме». Есть и другие уровни нормализации, однако для большинства приложений достаточно нормализовать базы данных до третьей нормальной формы.

Как и в случае со многими другими формальными правилами и спецификациями, обеспечить полное соответствие реальным ситуациям не всегда возможно. Как правило, для выполнения нормализации приходится создавать дополнительные таблицы, и некоторые клиенты считают это нежелательным. Собираясь нарушить одно из первых трех правил нормализации, убедитесь в том, что в приложении учтены все связанные с этим проблемы, такие как избыточность данных и несогласованные зависимости.

**Первая нормальная форма**. Отношение находится в 1НФ, если все его атрибуты являются простыми, все используемые домены должны содержать только скалярные значения. Не должно быть повторений строк в таблице.

**Вторая нормальная форма**. Отношение находится во 2НФ, если оно находится в 1НФ и каждый не ключевой атрибут неприводимо зависит от Первичного Ключа(ПК). Неприводимость означает, что в составе потенциального ключа отсутствует меньшее подмножество атрибутов, от которого можно также вывести данную функциональную зависимость.

**Третья нормальная форма**. Отношение находится в 3НФ, когда находится во 2НФ и каждый не ключевой атрибут не транзитивно зависит от первичного ключа. Проще говоря, второе правило требует выносить все не ключевые поля, содержимое которых может относиться к нескольким записям таблицы в отдельные таблицы. **Нормальная форма Бойса-Кодда (НФБК) (частная форма третьей нормальной формы)**. Определение 3НФ не совсем подходит для следующих отношений:

1) отношение имеет два или более потенциальных ключа;

2) два и более потенциальных ключа являются составными;

3) они пересекаются, т.е. имеют хотя бы один общий атрибут.

Для отношений, имеющих один потенциальный ключ (первичный), НФБК является 3НФ. Отношение находится в НФБК, когда каждая нетривиальная и неприводимая слева функциональная зависимость обладает потенциальным ключом в качестве детерминанта.

**Четвертая нормальная форма**. Отношение находится в 4НФ, если оно находится в НФБК и все нетривиальные многозначные зависимости фактически являются функциональными зависимостями от ее потенциальных ключей. В отношении R (A, B, C) существует **многозначная зависимость** R.A -> -> R.B в том и только в том случае, если множество значений B, соответствующее паре значений A и C, зависит только от A и не зависит от С.

**Пятая нормальная форма**. Отношения находятся в 5НФ, если оно находится в 4НФ и отсутствуют сложные зависимые соединения между атрибутами. Если «Атрибут\_1» зависит от «Атрибута\_2», а «Атрибут\_2» в свою очередь зависит от «Атрибута\_3», а «Атрибут\_3» зависит от «Атрибута\_1», то все три атрибута обязательно входят в один кортеж. Это очень жесткое требование, которое можно выполнить лишь при дополнительных условиях. На практике трудно найти пример реализации этого требования в чистом виде.

**Доменно-ключевая нормальная форма**. Переменная отношения находится в ДКНФ тогда и только тогда, когда каждое наложенное на неё ограничение является логическим следствием ограничений доменов и ограничений ключей, наложенных на данную переменную отношения. Ограничение домена – ограничение, предписывающее использовать для определённого атрибута значения только из некоторого заданного домена. Ограничение по своей сути является заданием перечня (или логического эквивалента перечня) допустимых значений типа и объявлением о том, что указанный атрибут имеет данный тип. Ограничение ключа – ограничение, утверждающее, что некоторый атрибут или комбинация атрибутов является потенциальным ключом. Любая переменная отношения, находящаяся в ДКНФ, обязательно находится в 5НФ. Однако не любую переменную отношения можно привести к ДКНФ.

**Шестая нормальная форма**. Переменная отношения находится в шестой нормальной форме тогда и только тогда, когда она удовлетворяет всем нетривиальным зависимостям соединения. Из определения следует, что переменная находится в 6НФ тогда и только тогда, когда она неприводима, то есть не может быть подвергнута дальнейшей декомпозиции без потерь. Каждая переменная отношения, которая находится в 6НФ, также находится и в 5НФ.

Идея «декомпозиции до конца» выдвигалась до начала исследований в области хронологических данных, но не нашла поддержки. Однако для хронологических баз данных максимально возможная декомпозиция позволяет бороться с избыточностью и упрощает поддержание целостности базы данных.

Для хронологических баз данных определены U\_операторы, которые распаковывают отношения по указанным атрибутам, выполняют соответствующую операцию и упаковывают полученный результат. В данном примере соединение проекций отношения должно производится при помощи оператора U\_JOIN.

# **Проектирование реляционных баз данных.**

Проектирование баз данных информационных систем является достаточно трудоемкой задачей. Оно осуществляется на основе формализации структуры и процессов предметной области, сведения о которой предполагается хранить в БД. Различают **концептуальное** и **схемно-структурное** проектирование.

**Концептуальное проектирование** БД ИС является в значительной степени эвристическим процессом. Адекватность построенной в его рамках инфологической модели предметной области проверяется опытным путем, в процессе функционирования ИС.

Перечислим **этапы** концептуального проектирования:

• изучение предметной области для формирования общего представления о ней;

• выделение и анализ функций и задач разрабатываемой ИС;

• определение основных объектов-сущностей предметной области и отношений между ними;

• формализованное представление предметной области.

При проектировании схемы реляционной БД можно выделить следующие **процедуры**:

• определение перечня таблиц и связей между ними;

• определение перечня полей, типов полей, ключевых полей каждой таблицы (схемы таблицы), установление связей между таблицами через внешние ключи;

• установление индексирования для полей в таблицах;

• разработка списков (словарей) для полей с перечислительными данными;

 установление ограничений целостности для таблиц и связей;

• нормализация таблиц, корректировка перечня таблиц и связей.

Проектирование БД осуществляется на физическом и логическом уровнях. Проектирование на физическом уровне реализуется средствами СУБд и зачастую автоматизировано.

Логическое проектирование заключается в определении числа и структуры таблиц, разработке запросов к БД, отчетных документов, создании форм для ввода и редактирования данных в БД и т. д.

Одной из важнейших задач логического проектирования БД является структуризация данных. Выделяют следующие подходы к проектированию структур данных [14]:

• объединение информации об объектах-сущностях в рамках одной таблицы (одного отношения) с последующей декомпозицией на несколько взаимосвязанных таблиц на основе процедуры нормализации отношений;

• формулирование знаний о системе (определение типов исходных данных и взаимосвязей) и требований к обработке данных, получение с помощью CASE-системы готовой схемы БД или даже готовой прикладной информационной системы;

• осуществление системного анализа и разработка структурных моделей.

# **Язык SQL.**

**Структурированный язык запросов SQL** основан на реляционном исчислении с *переменными кортежами.*Язык имеет несколько стандартов, наиболее распространенными из которых являются SQL-89 и SQL-92.

Язык SQL предназначен для выполнения операций над таблицами (создание, удаление, изменение структуры) и над данными таблиц (выборка, изменение, добавление и удаление), а также некоторых сопутствующих операций. SQL является *непроцедурным*языком и не содержит операторов управления, организации подпрограмм, ввода-вывода и т. п. В связи с этим SQL автономно не используется, обычно он погружен в среду встроенного языка программирования СУБД (например, FoxPro СУБД Visual FoxPro, ObjectPAL СУБД Paradox, Visual Basic for Applications СУБД Access).

В современных СУБД с интерактивным интерфейсом можно создавать запросы, используя другие средства, например, QBE. Однако применение SQL зачастую позволяет повысить эффективность обработки данных в базе. Например, при подготовке запроса в среде Access можно перейти из окна Конструктора запросов (формулировки запроса по образцу на языке QBE) в окно с эквивалентным оператором SQL. Подготовку нового запроса путем редактирования уже имеющегося в ряде случае проще выполнить путем изменения оператора SQL. В различных СУБД состав операторов SQL может несколько отличаться.

Язык SQL преобладает функциями полноценного языка разработки, а ориентирован на доступ к данным, поэтому его включают в состав средств разработки программ. В этом случае его называют *встроенным SQL.*Стандарт языка SQL поддерживают современные реализации следующих языков программирования: PL/1, Ada, С, COBOL, Fortran, M U M P S и Pascal.

В специализированных системах разработки приложений типа клиент-сервер среда программирования, кроме того, обычно дополнена коммуникационными средствами (установление и разъединение соединений с серверами БД, обнаружение и обработка возникающих в сети ошибок и т. д.), средствами разработки пользовательских интерфейсов, средствами проектирования и отладки.

Различают два основных метода использования встроенного SQL: статический и динамический.

При *статическом*использовании языка ***(статический SQL)*** в тексте программы имеются вызовы функций языка SQL, которые жестко включаются в выполняемый модуль после компиляции. Изменения в вызываемых функциях могут быть на уровне отдельных параметров вызовов с помощью переменных языка программирования.

При *динамическом*использовании языка (***динамический SQL)***предполагается динамическое построение вызовов SQL-функций и интерпретация этих вызовов, например, обращение к данным удаленной базы, в ходе выполнения программы. Динамический метод обычно применяется в случаях, когда в приложении заранее неизвестен вид SQL-вызова и он строится в диалоге с пользователем.

Основным назначением языка SQL (как и других языков для работы с базами данных) является подготовка и выполнение запросов. В результате выборки данных из одной или нескольких таблиц может быть получено множество записей, называемое *представлением.*

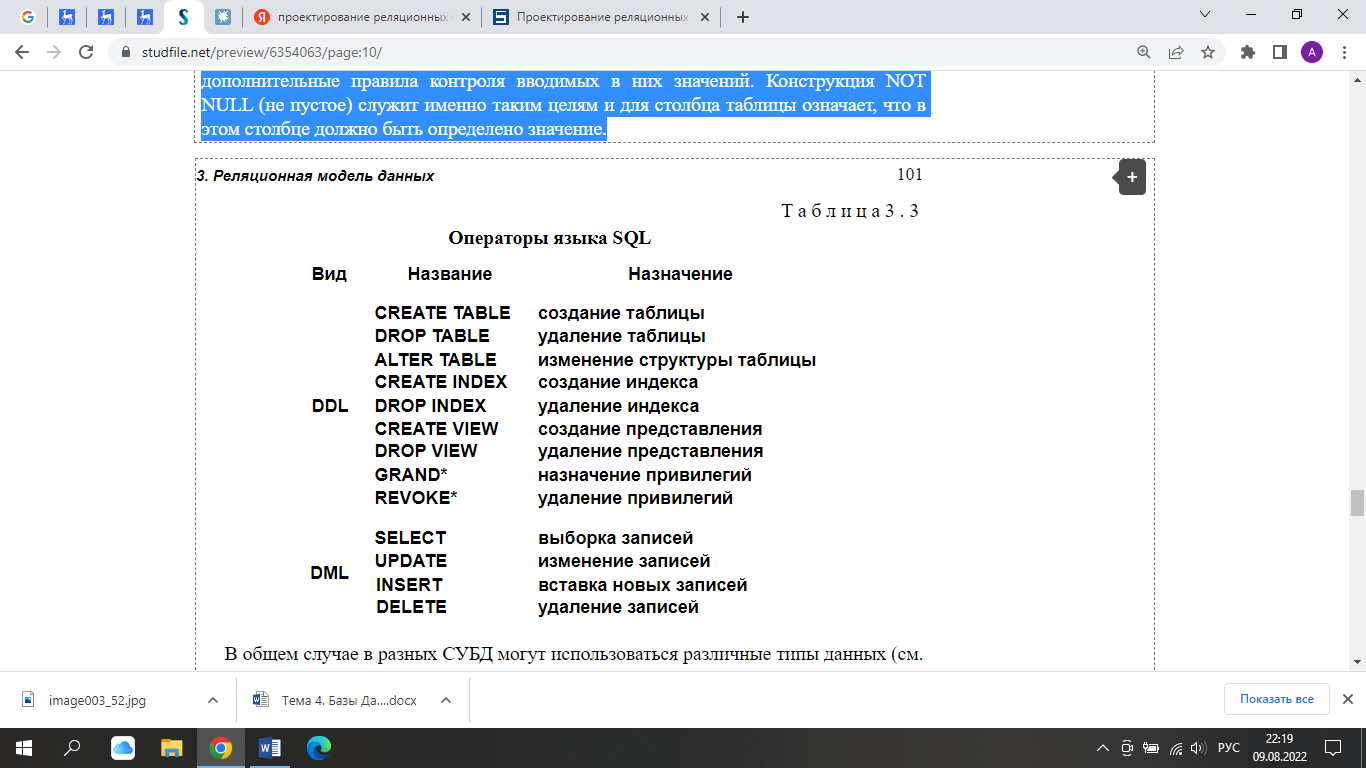
***Представление***по существу является таблицей, формируемой в результате выполнения запроса. Можно сказать, что оно является разновидностью хранимого запроса. По одним и тем же таблицам можно построить несколько представлений. Само представление описывается путем указания идентификатора представления и запроса, который должен быть выполнен для его получения.

Для удобства работы с представлениями в язык SQL введено понятие курсора. ***Курсор***представляет собой своеобразный указатель, используемый для перемещения по наборам записей при их обработке.

Описание и использование курсора в языке SQL выполняется следующим образом. В описательной части программы выполняют связывание переменной типа курсор (CURSOR) с оператором SQL (обычно с оператором SELECT). В выполняемой части программы производится открытие курсора (OPEN <имя курсора>), перемещение курсора по записям (FETCH <имя курсора>...), сопровождаемое соответствующей обработкой, и, наконец, закрытие курсора (CLOSE <имя курсора>).

***Основные операторы языка.*** Опишем минимальное подмножество языка SQL, опираясь на его реализацию в стандартном интерфейсе ODBC (Open Database Connectivity — совместимость открытых баз данных) фирмы Microsoft.

Операторы языка SQL можно условно разделить на два подъязыка: язык определения данных (Data Definition Language — DDL) и язык манипулирования данными (Data Manipulation Language — DML). Основные операторы языка SQL представлены в табл. 3.3.



Рассмотрим формат и основные возможности важнейших операторов, за исключением специфических операторов, отмеченных в таблице символом «\*». Несущественные операнды и элементы синтаксиса (например, принятое во многих системах программирования правило ставить «;» в конце оператора) будем опускать.

1. Операторт ***создания таблицы*** имеет формат вида:

CREATE TABLE <имя таблицы>

(<имя столбца> <тип данных> [NOT NULL] [,<имя столбца> <тип данных> [NOT NULL]]... )

Обязательными операндами оператора являются имя создаваемой таблицы и имя хотя бы одного столбца (поля) с указанием типа данных, хранимых в этом столбце.

При создании таблицы для отдельных полей могут указываться некоторые дополнительные правила контроля вводимых в них значений. Конструкция NOT NULL (не пустое) служит именно таким целям и для столбца таблицы означает, что в этом столбце должно быть определено значение.

В общем случае в разных СУБД могут использоваться различные типы данных (см. подраздел 2.7). В интерфейсе ODBC поддерживаются свои стандартные типы данных, например, символьные (SQL \_ CHAR, SQL\_VARCHAR, SQL LONGVARCHAR) и др. При работе с БД некоторой СУБД посредством интерфейса ODBC выполняется автоматическое преобразование стандартных типов данных, поддерживаемых интерфейсом, в типы данных источников и обратно. При необходимости обмен данными между программой и источником данных может вестись без преобразования — во внутреннем формате данных источника.

1. Оператор ***изменения структуры таблицы***имеет формат вида:

ALTER TABLE <имя таблицы>

( {ADD, MODIFY, DROP} <имя столбца> [<тип данных>] [NOT NULL]

[,{ADD, MODIFY, DROP} <имя столбца> [<тип данных>] [NOT NULL]]...)

Изменение структуры таблицы может состоять в добавлении (ADD), изменении (MODIFY) или удалении (DROP) одного или нескольких столбцов таблицы. Правила записи оператора ALTER TABLE такие же, как и оператора CREATE TABLE. При удалении столбца указывать <тип данных> не нужно.

1. Оператор ***удаления таблицы***имеет формат вида:

DROP TABLE <имя таблицы>

Оператор позволяет удалить имеющуюся таблицу. Например, для удаления таблицы с именем items достаточно записать оператор вида:

DROP TABLE items.

1. Оператор ***создания индекса***имеет формат вида:

CREATE [UNIQUE] INDEX <имя индекса> ON  
 <имя таблицы>

(<имя столбца> [ ASC | DESC ] [,<имя столбца> [ ASC | DESC ]... )

Оператор позволяет создать индекс для одного или нескольких столбцов заданной таблицы с целью ускорения выполнение запросных и поисковых операций с таблицей. Для одной таблицы можно создать несколько индексов.

Задав необязательную опцию UNIQUE, можно обеспечить уникальность значений во всех указанных в операторе столбцах. По существу, создание индекса с указанием признака UNIQUE означает определение ключа в созданной ранее таблице.

При создании индекса можно задать порядок автоматической сортировки значений в столбцах — в порядке возрастания ASC (по умолчанию), или в порядке убывания DESC. Для разных столбцов можно задавать различный порядок сортировки.

1. Оператор ***удаления индекса***имеет формат вида:

DROP INDEX <имя индекса>

Этот оператор позволяет удалять созданный ранее индекс с соответствующим именем. Так, например, для уничтожения индекса main\_indx к таблице emp достаточно записать оператор DROP INDEX main\_indx.

1. Оператор ***создания представления*** имеет формат вида:

CREATE VIEW <имя представления>  
 [(<имя столбца> [,<имя столбца>]...)]

AS <оператор SELECT>

Данный оператор позволяет создать представление. Если имена столбцов в представлении не указываются, то будут использоваться имена столбцов из запроса, описываемого соответствующим оператором SELECT.

1. Оператор ***удаления представления***имеет формат вида:

DROP VIEW <имя представления>

Оператор позволяет удалить созданное ранее представление. Заметим, что при удалении представления таблицы, участвующие в запросе, удалению не подлежат. Удаление представления производится оператором вида:

DROP VIEW герг.

1. Оператор ***выборки записей***имеет формат вида:

SELECT [ALL | DISTINCT]  
 <список данных> FROM  
 <список таблиц>

WHERE <условие выборки>]

[GROUP BY <имя столбца> [,<имя столбца>] ... ] [HAVING <условие поиска>]

[ORDER BY <спецификация> [,<спецификация>] ...]

# **Введение в MS SQL Server.**

SQL Server является одной из наиболее популярных систем управления базами данных (СУБД) в мире. Данная СУБД подходит для самых различных проектов: от небольших приложений до больших высоконагруженных проектов. SQL Server был создан компанией Microsoft. Первая версия вышла в 1987 году. А текущей версией является версия 2019, которая вышла в 2019 году. SQL Server долгое время был исключительно системой управления базами данных для Windows, однако начиная с версии 16 эта система доступна и на Linux.

**SQL Server характеризуется такими особенностями как**:

* Производительность. SQL Server работает очень быстро.
* Надежность и безопасность. SQL Server предоставляет шифрование данных.
* Простота. С данной СУБД относительно легко работать и вести администрирование.

Центральным аспектом в MS SQL Server, как и в любой СУБД, является база данных. **База данных** представляет хранилище данных, организованных определенным способом. Нередко физически база данных представляет файл на жестком диске, хотя такое соответствие необязательно. Для хранения и администрирования баз данных применяются системы управления базами данных (database management system) или СУБД (DBMS). И как раз MS SQL Server является одной из такой СУБД.

Для организации баз данных MS SQL Server использует реляционную модель. Эта модель баз данных была разработана еще в 1970 году Эдгаром Коддом. А на сегодняшний день она фактически является стандартом для организации баз данных.

Реляционная модель предполагает хранение данных в виде таблиц, каждая из которых состоит из строк и столбцов. Каждая строка хранит отдельный объект, а в столбцах размещаются атрибуты этого объекта.

Для идентификации каждой строки в рамках таблицы применяется первичный ключ (primary key). В качестве первичного ключа может выступать один или несколько столбцов. Используя первичный ключ, мы можем ссылаться на определенную строку в таблице. Соответственно две строки не могут иметь один и тот же первичный ключ.

Через ключи одна таблица может быть связана с другой, то есть между двумя таблицами могут быть организованы связи. А сама таблица может быть представлена в виде отношения ("relation").

Для взаимодействия с базой данных применяется язык SQL (Structured Query Language). Клиент (например, внешняя программа) отправляет запрос на языке SQL посредством специального API. СУБД должным образом интерпретирует и выполняет запрос, а затем посылает клиенту результат выполнения.

Изначально язык SQL был разработан в компании IBM для системы баз данных, которая называлась System/R. При этом сам язык назывался SEQUEL (Structured English Query Language). Хотя в итоге ни база данных, ни сам язык не были впоследствии официально опубликованы, по традиции сам термин SQL нередко произносят как "сиквел".

Впоследствии стали появляться другие системы баз данных, которые использовали SQL. В итоге в 1989 году Американский Национальный Институт Стандартов (ANSI) кодифицировал язык и опубликовал его первый стандарт. После этого стандарт периодически обновлялся и дополнялся. Последнее его обновление состоялось в 2011 году. Но несмотря на наличие стандарта нередко производители СУБД используют свои собственные реализации языка SQL, которые немного отличаются друг от друга.

Выделяются две разновидности языка SQL: PL-SQL и T-SQL.

**PL-SQL** используется в таких СУБД как Oracle и MySQL.

**T-SQL (Transact-SQL)** применяется в SQL Server.

# **Контроль целостности данных в MS SQL Server.**

Главная особенность SQL-технологий наличие у сервера СУБД специальных средств контроля целостности данных, не зависящих от клиентских программ и привязанных непосредственно к таблицам.

К структурам контроля целостности данных относятся ограничители (constraint), которые привязаны к столбцам и триггеры (trigger), которые могут быть привязаны как к столбцам, так и к строкам в таблице.

**Ограничители** — это элементарные проверки или условия, которые выполняются для операций вставки и модификации значения столбца. Если данная проверка не проходит или условие не выполняется, то вставка или модификация отменяется, а в программу клиента передается ошибка.

SQL-серверы, как правило, поддерживают следующие ограничители:

NOT NULL - проверка на непустое значение.

UNIQUE - проверка на уникальность.

PRIMARY KEY - первичный ключ.

FOREIGN KEY - внешний ключ.

REFERENCES - указатель ссылки (или родительский ключ).

CHECK - проверка фиксированного условия.

Обычно ограничители задаются при создании таблиц. Но в дальнейшем их можно изменять, удалять или временно запрещать при помощи соответствующих команд СУБД.

**Триггеры** – это сохраненная откомпилированная процедура, которая связана с определенной таблицей. Триггеры, в отличие от ограничителей, могут выполнять сколь угодно сложные манипуляции над данными. Помимо операций модификации и вставки, триггеры могут срабатывать и при удалении данных из таблицы. Можно также задавать порядок срабатывания триггера относительно операции, т.е. выполниться ли триггер перед операцией вставки/модификации/удаления значения из столбца (или всей строки) или непосредственно после такой операции.

# **Язык Transact-SQL.**

**Transact-SQL ( T-SQL )** - это процедурное расширение Microsoft и Sybase для SQL (структурированный язык запросов), используемый для взаимодействия с реляционными базами данных . T-SQL расширяет стандарт SQL для включения процедурного программирования, локальных переменных, различных вспомогательных функций для обработки строк, обработки даты, математики, а также изменений в операторах DELETE и UPDATE

Transact-SQL занимает центральное место в использовании Microsoft SQL Server. Все приложения, которые обмениваются данными с экземпляром SQL Server, делают это путем отправки операторов Transact-SQL на сервер, независимо от пользовательского интерфейса приложения.

Хранимые процедуры в SQL Server - это исполняемые серверные процедуры. Преимуществом хранимых процедур является возможность передачи параметров.

**T-SQL (Transact-SQL)** применяется в SQL Server.

Собственно, поэтому в рамках текущего руководства будет рассматриваться именно T-SQL.

В зависимости от задачи, которую выполняет команда T-SQL, он может принадлежать к одному из следующих типов:

**DDL** (Data Definition Language / Язык определения данных). К этому типу относятся различные команды, которые создают базу данных, таблицы, индексы, хранимые процедуры и т.д. В общем определяют данные.

В частности, к этому типу мы можем отнести следующие команды:

**CREATE**: создает объекты базы данных (саму базу даных, таблицы, индексы и т.д.)

**ALTER**: изменяет объекты базы данных

**DROP**: удаляет объекты базы данных

**TRUNCATE**: удаляет все данные из таблиц

**DML** (Data Manipulation Language / Язык манипуляции данными). К этому типу относят команды по выбору данных, их обновление, добавление, удаление - в общем все те команды, с помощью которыми мы можем управлять данными.

К этому типу относятся следующие команды:

**SELECT**: извлекает данные из БД

**UPDATE**: обновляет данные

**INSERT**: добавляет новые данные

**DELETE**: удаляет данные

**DCL** (Data Control Language / Язык управления доступа к данным). К этому типу относят команды, которые управляют правами по доступу к данным. В частности, это следующие команды:

**GRANT**: предоставляет права для доступа к данным

**REVOKE**: отзывает права на доступ к данным.

# **Хранимые процедуры.**

**Хранимая процедура в SQL Server** — это группа из одной инструкции Transact-SQL или нескольких, или ссылка на метод среды CLR Microsoft .NET Framework. Процедуры аналогичны конструкциям в других языках программирования, поскольку обеспечивают следующее:

* обрабатывают входные параметры и возвращают вызывающей программе значения в виде выходных параметров;
* содержат программные инструкции, которые выполняют операции в базе данных, включая вызов других процедур;
* возвращают значение состояния вызывающей программе, таким образом передавая сведения об успешном или неуспешном завершении (и причины последнего).

**Преимущества хранимых процедур.**

1. Снижение сетевого трафика между клиентами и сервером.  
   Команды в процедуре выполняются как один пакет кода. Это позволяет существенно сократить сетевой трафик между сервером и клиентом, поскольку по сети отправляется только вызов на выполнение процедуры. Без инкапсуляции кода, предоставляемой процедурой, по сети бы пришлось пересылать все отдельные строки кода.
2. Большая безопасность.  
   Многие пользователи и клиентские программы могут выполнять операции с базовыми объектами базы данных посредством процедур, даже если у них нет прямых разрешений на доступ к базовым объектам. Процедура проверяет, какие из процессов и действий могут выполняться, и защищает базовые объекты базы данных. Это устраняет необходимость предоставлять разрешения на уровне индивидуальных объектов и упрощает формирование уровней безопасности.
3. Повторное использование кода.  
   Если какой-то код многократно используется в операции базы данных, то отличным решением будет произвести его инкапсуляцию в процедуры. Это устранит необходимость излишнего копирования того же кода, снизит уровень несогласованности кода и позволит осуществлять доступ к коду любым пользователям или приложениям, имеющим необходимые разрешения.
4. Более легкое обслуживание.  
   Если клиентские приложения вызывают процедуры, а операции баз данных остаются на уровне данных, то для внесения изменений в основную базу данных будет достаточно обновить только процедуры. Уровень приложения остается незатронутым изменениями в схемах баз данных, связях или процессах.
5. Повышение производительности.  
   По умолчанию компиляция процедуры и создание плана выполнения, используемого для последующих выполнений, производится при ее первом запуске Поскольку обработчику запросов не нужно создавать новый план, обычно обработка процедуры занимает меньше времени.

Типы хранимых процедур:

**Пользовательские процедуры** могут быть созданы в пользовательской базе данных или любых системных базах данных, за исключением базы данных **Resource** . Процедура может быть разработана либо на языке Transact-SQL, либо как ссылка на метод Microsoft .NET Framework среды CLR.

**Временные процедуры** — это один из видов пользовательских процедур. Временные процедуры схожи с постоянными процедурами, за исключением того, что они хранятся в базе данных **tempdb**. Существует два вида временных процедур: локальные и глобальные. Они отличаются друг от друга именами, видимостью и доступностью. Имена локальных временных процедур начинаются с одного знака диеза (#); они видны только текущему соединению пользователя и удаляются, когда закрывается соединение. Имена глобальных временных процедур начинаются с двух знаков диеза (##); они видны любому пользователю и удаляются после окончания последнего сеанса, использующего процедуру.

**Системные функции**. Системные процедуры включены в SQL Server. Физически они хранятся во внутренней скрытой базе данных **Resource** . Логически они отображаются в схеме **sys** каждой системной и пользовательской базы данных. В дополнение к этому, база данных **msdb** также содержит системные хранимые процедуры в схеме **dbo** . Эти процедуры используются для планирования предупреждений и заданий. Поскольку названия системных процедур начинаются с префикса **sp\_** , этот префикс не рекомендуется использовать при создании пользовательских процедур.

**Расширенные пользовательские процедуры** позволяют создавать внешние подпрограммы на языках программирования, таких как С. Они представляют собой библиотеки DLL, которые могут динамически загружаться и выполняться экземпляром SQL Server.

# **Функции.**

# **Представления.**

# **Триггеры.**

# **Курсоры.**

# **Оперативный анализ данных.**