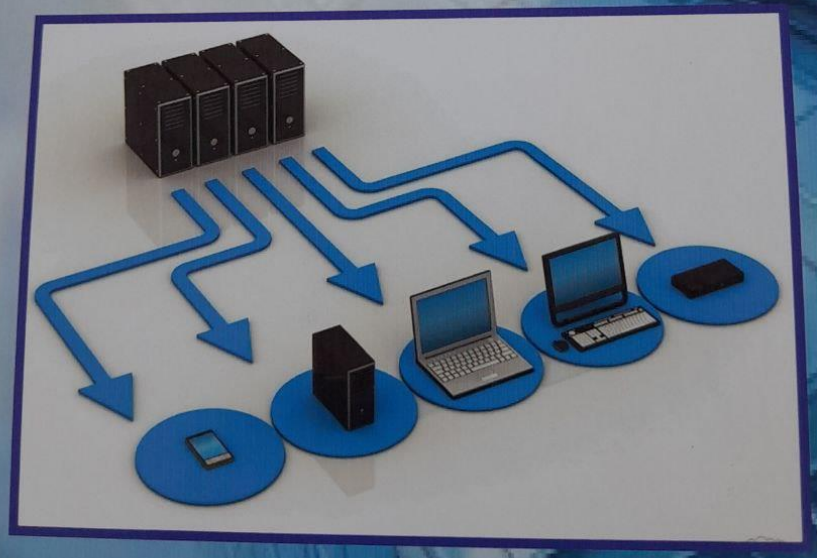
Х.Ж. Рахимбоев, У.Ф. Зарипов

БАЗЫ ДАННЫХ



МИНИСТЕРСТВО ЦИФРОВЫХ ТЕХНОЛОГИЙ
РЕСПУБЛИКИ УЗБЕКИСТАН

УРГЕНЧСКИЙ ФИЛИАЛ ТАШКЕНТСКОГО
УНИВЕРСИТЕТА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ
ИМЕНИ МУХАММАДА АЛЬ-ХОРЕЗМИ

Х.Ж. Рахимбоев, У.Ф. Зарипов

БАЗЫ ДАННЫХ

Учебное пособие

УДК: 004.6(075.8)
ББК: 32.973я7
Р 271

Рахимбоев Х. Ж.
Базы данных. Учебное пособие / Зарипов У.Ф. /. – Ташкент: "METODIST NASHRIYOTI", 2024. – 192 стр.

В пособии дается общий обзор различных типов баз данных и соответствующих методов их использования. Описываются реляционная модель баз данных, основные конструкции языка SQL, методы проектирования структуры баз данных, основанные как на принципах нормализации, так и на использовании модели «сущность-связь». Приведены основные понятия о транзакциях в базах данных. В заключение рассмотрен общий подход к технологиям «клиент-сервер».

Предназначено для студентов, обучающихся по направлениям 350400 – проф. образования в сфере ИКТ. 5330501- Компьютерный инжиниринг («Компьютерный инжиниринг», «ИТ-Сервис»). 330600 – Программный инжиниринг, 5330300 – Информационная безопасность, Телекоммуникационные технологии., очной формы обучения.

Рецензенты:

М.С.Шарипов - кандидат технических наук, доцент; кафедра информационные технологии Ургенчского государственного университета.

ISBN 978-9910-03-117-5

ПРЕДИСЛОВИЕ

Умение грамотно работать с современными базами данных является одним из ключевых требований к любому специалисту в области компьютерных технологий. Важную роль в освоении технологий баз данных играет понимание их теоретических основ. Предлагаемое учебное пособие нацелено именно на ознакомление студентов с фундаментальными понятиями баз данных. В нем дается общий обзор различных типов баз данных и соответствующих методов их использования. Особое внимание уделяется наиболее распространенным реляционным базам данных. Описываются основные конструкции языка SQL, являю- щегося фактическим стандартом работы с реляционными базами данных. Приведены методы проектирования структуры баз данных, основанные как на принципах нормализации, так и на использовании модели «сущность-связь». Отдельная глава посвящена описанию транзакций, являющихся основным средством обеспечения надежной работы с базами данных. В заключение рассматривается общий подход к технологиям «XML и база данных» и системы распределенной обработки данных.

1. БАЗЫ И БАНКИ ДАННЫХ. ОСНОВНОЕ ПОНЯТИЯ И ОПИСАНИЯ. ТРЕБОВАНИЕ К БАЗАМ ДАННЫХ

1.1. Понятия информации

Под информацией понимают любые сведения о каком-либо событии, сущности, процессе и т.п., являющиеся объектом некоторых операций: восприятия, передачи, преобразования, хранения или использования.

Понятие об информации сложилось у человека давно. Она используется во всех областях человеческой деятельности, любая взаимосвязь и координация работ возможны только благодаря информации. Человек создал естественные информационные системы, поскольку существовала насущная потребность снабжать производство информацией, необходимой при контроле и принятии решений; научился собирать эту информацию, обрабатывать и передавать ее по назначению.

Перед тем как определить понятие «данные», представим следующую абстрактную ситуацию. Имеются некоторая система, информация о которой представляет интерес, и наблюдатель, способный воспринимать состояния системы и в определенной форме фиксировать их в своей памяти (никаких других действий наблюдатель не выполняет). В этом случае считают, что в памяти наблюдателя находятся данные, описывающие состояние системы. Таким наблюдателем в общем случае выступает информационная система.

Итак, *данные* можно определить как информацию, фиксированную в определенной форме, пригодной для последующей обработки, хранения и передачи.

Соответственно двум понятиям «информация» и «данные» различают два аспекта рассмотрения вопросов — инфологический и даталогический.

В инфологическом аспекте рассматриваются вопросы, связанные со смысловым содержанием данных независимо от способов их представления в памяти системы. На этапе инфологического проектирования информационной системы решаются следующие вопросы:

1. О каких объектах или явлениях реального мира требуется накапливать и обрабатывать информацию в системе?

2. Какие их основные характеристики и взаимосвязи между собой будут учитываться?

3. Какие вводимые в информационную систему понятия об объектах и явлениях, их характеристиках и взаимосвязях требуют уточнения?

Таким образом, на этапе инфологического проектирования выделяется часть реального мира, определяющая информационные потребности системы, т.е. ее предметная область.

В даталогическом аспекте рассматриваются вопросы представления данных в памяти информационной системы. При даталогическом проектировании системы, исходя из возможностей имеющихся средств

восприятия, хранения и обработки информации, разрабатываются соответствующие формы представления информации в системе посредством данных, а также приводятся модели и методы представления и преобразования данных, формируются правила смысловой интерпретации данных.

Данные соответствуют зарегистрированным фактам об объектах или явлениях реального мира. Чтобы в дальнейшем использовать данные, требуется их смысловое содержание — семантика данных. Поэтому в информационной системе должны быть сформулированы правила их смысловой интерпретации.

1.2. Автоматизированные информационные системы и банки данных

Длительное время различные направления автоматизированных информационных систем (АИС) развивались независимо, и поэтому в настоящее время нет единой трактовки и устоявшейся их классификации.

В 60-е годы была осознана роль информации как важнейшего ресурса любой организации, предприятия. Началась разработка АИС различного назначения, совершенствовались различные программные процедуры и средства вычислительной техники для обработки данных, наращивалась память ЭВМ, развивались средства телекоммуникаций. Работы по созданию АИС в нашей стране велись в двух направлениях:

1) разработка АИС как первой очереди автоматизированных систем управления (АСУ);

2) разработка автоматизированных систем научно-технической информации (АСИТИ).

В первом направлении выделялись АСУ различных уровней: АСУТП — для автоматизации технологических процессов, АСУП — для автоматизации организационного управления предприятием или организацией, ОАСУ — отраслевые автоматизированные системы управления, ОГАС — общегосударственная автоматизированная система.

Второе направление было связано с обеспечением научно-технической информацией практически всех видов народнохозяйственной деятельности. В нашей стране был принят единый порядок разработки общегосударственной АСНТИ. В ее структуре были предусмотрены общегосударственные, отраслевые и региональные органы, отделы и бюро научно-технической информации (НТИ, ОНТИ и БТИ соответственно) на предприятиях, в научно-исследовательских институтах и других организациях.

Банк данных (БнД) — это АИС, включающая в свой состав комплекс специальных методов и средств (математических, информационных программных, языковых, организационных и технических) для поддержани динамической информационной модели предметной области с цель обеспечения информационных запросов пользователей.

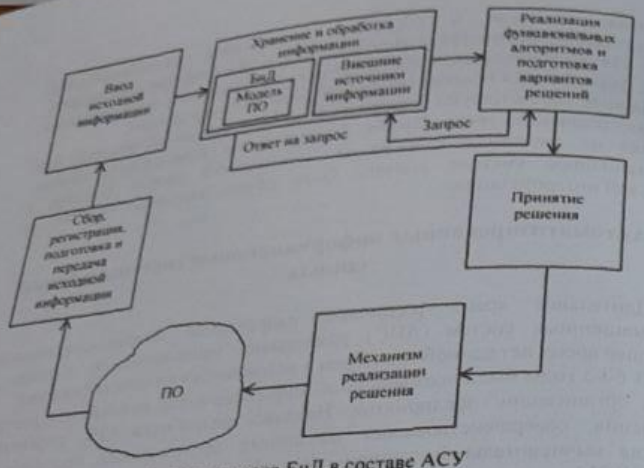


Рис. 1.1. Роль и место БнД в составе АСУ

Предметная область (ПО) — это область применения конкретного БнД. Различают БнД, применяемые в сфере управления предприятиями и организациями, транспортом, в медицине, научных исследованиях и т.д.

Банк данных выступает в роли специальной обеспечивающей подсистемы в составе АСУ различного профиля (рис. 1.1).

Задача поддержания информационной модели в необходимом состоянии требует, чтобы в БнД выполнялись операции хранения и модификации (последняя представляет собой совокупность операций «включить», «удалить», «изменить данные») информационной модели в соответствии с возникающими изменениями в состоянии объектов ПО. Кроме того, с развитием АС видоизменяется состав объектов ПО и связи между ними, что также должно найти отражение в соответствующих изменениях информационной модели. В АС используется самая разнообразная по смысловому содержанию информация, представленная в различных кодах. Поэтому организация БнД должна быть достаточно гибкой, чтобы обеспечивать использование информации различных видов и изменять при необходимости структуру хранимой информации.

Услугами БнД обычно пользуется большое число пользователей, поэтому в нем предусматривается специальное средство приведения всех запросов к единой терминологии — словарь данных. Кроме того, существуют специальные методы эквивалентных грамматических преобразований

запросов для построения оптимальных процедур их обработки, а также специальные методы доступа к одним и тем же данным различных пользователей при совпадении во времени поступивших запросов. Как правило, со стороны внешних пользователей к БнД предъявляют следующие требования:

1) удовлетворять актуальным информационным потребностям внешних пользователей, обеспечивать возможность хранения и модификации больших объемов многоаспектной информации, удовлетворять выявленным и вновь возникающим потребностям внешних пользователей;

2) обеспечивать заданный уровень достоверности хранимой информации и ее непротиворечивость;

3) обеспечивать доступ к данным только пользователям с соответствующими полномочиями;

4) обеспечивать возможность поиска информации по произвольной группе признаков;

5) удовлетворять заданным требованиям по производительности при обработке запросов;

6) иметь возможность реорганизации и расширения при изменении границ ПО;

7) обеспечивать выдачу информации пользователю в различной форме;

8) обеспечивать простоту и удобство обращения внешних пользователей за информацией;

9) обеспечивать возможность одновременного обслуживания большого числа внешних пользователей и т.п.

Стремление к максимальному удовлетворению перечисленных требований приводит к необходимости решения вопроса о централизации управления данными, имеющей ряд преимуществ.

1. Сокращение избыточности хранимых данных. Может быть обеспечена минимально необходимая (например, только для обеспечения требуемой производительности системы) избыточность (дублирование) хранимых данных. При установлении факта использования несколькими программами одинаковых данных, такие данные интегрируют и хранят в единственном экземпляре. В дальнейшем их используют во всех соответствующих прикладных программах.

2. Устранение противоречивости хранимых данных. Следствием устранения избыточности данных является устранение возможности возникновения противоречивости одних и тех же данных, хранимых в различных файлах.

3. Многоаспектное использование данных. Централизованное управление позволяет в полной мере решать такой вопрос, как обеспечение новых приложений за счет уже имеющихся данных, т.е. обеспечивается реализация принципа однократного ввода и многократного (многоаспектного) использования данных.

4. Комплексная оптимизация. В максимальной степени устраняются противоречивые требования. Например, на основе анализа требований пользователей можно выбрать такие структуры хранения данных, которые обеспечат наилучшее обслуживание в целом.

5. Обеспечение возможности стандартизации. Например, в представлении данных это упрощает эксплуатацию БнД, обмен данными с другими АС, облегчает выполнение процедур контроля и восстановления данных.

6. Обеспечение возможности санкционированного доступа к данным. Интеграция (объединение) данных приводит к тому, что используемые различными пользователями, они могут пересекаться. В этих условиях особенно важно наличие механизма защиты данных от несанкционированного доступа к ним, т.е. доступ к определенным группам данных должен разрешаться только пользователям с соответствующими полномочиями.

Предоставляя определенные преимущества, централизованное управление данными выдвигает на первый план проблему обеспечения независимости от них прикладных программ. Эта проблема существовала и до появления БнД, так как ее решение обеспечивало снижение затрат ручного труда при написании и корректировке программ. С появлением БнД потребовалось кардинальное решение этой проблемы, поскольку при интеграции данных и оптимизации структур хранения с целью улучшения характеристик процессов обслуживания запросов пользователей требуется изменять хранимое представление данных и методы доступа к ним. Обеспечение независимости прикладных программ от изменений в хранимых данных становится насущной необходимостью. В противном случае требуется выполнять трудоемкие ручные операции по внесению соответствующих изменений в прикладные программы.

Рассматривая данные как один из ресурсов АС, можно отметить, что БнД централизованно управляет этим ресурсом в интересах всей системы. Наличие централизованного управления данными — главная отличительная черта БнД. Таким образом, БнД — это информационная система, реализующая централизованное управление данными в интересах всех пользователей АС, в состав которой она входит.

Банки данных возникли в связи с потребностью в интеграции данных. Дальнейшее продвижение в этом направлении потребовало решения проблемы интеграции и процессов обработки, что привело к появлению банков знаний (БнЗ).

Информация в современном мире превратилась в один из наиболее жных ресурсов, а информационные системы (ИС) стали необходимым струментом практически во всех сферах деятельности.

Разнообразие задач, решаемых с помощью ИС, привело к появлению жества разнотипных систем, отличающихся принципами построения и женными в них правилами обработки информации.

Информационные системы можно **классифицировать** по целому ряду различных признаков. В основу рассматриваемой классификации положены наиболее существенные признаки, определяющие функциональные возможности и особенности построения современных систем. В зависимости от объема решаемых задач, используемых технических средств, организации функционирования, информационные системы делятся на ряд групп (классов) (рис1.1).

По типу хранимых данных ИС делятся на фактографические и документальные. Фактографические системы предназначены для хранения и обработки структурированных данных в виде чисел и текстов. Над такими данными можно выполнять различные операции. В документальных системах информация представлена в виде документов, состоящих из наименований, описаний, рефератов и текстов. Поиск по неструктурированным данным осуществляется с использованием семантических признаков. Отобранные документы предоставляются пользователю, а обработка данных в таких системах практически не производится.

Рис. 1.1. Классификация информационных систем

Основываясь на **степени автоматизации информационных процесс** в системе управления фирмой, информационные системы делятся на ручны автоматические и автоматизированные.

Ручные ИС характеризуются отсутствием современных технических средств переработки информации и выполнением всех операций человеком.

В автоматических ИС все операции по переработке информации выполняются без участия человека.

Автоматизированные ИС предполагают участие в процессе обработки информации и человека, и технических средств, причем главная роль в выполнении рутинных операций обработки данных отводится компьютеру. Именно этот класс систем соответствует современному представлению понятия "информационная система".

В зависимости от **характера обработки** данных ИС делятся на информационно-поисковые и информационно-решающие.

Информационно-поисковые системы производят ввод, систематизацию, хранение, выдачу информации по запросу пользователя без сложных преобразований данных. (Например, ИС библиотечного обслуживания, резервирования и продажи билетов на транспорте, бронирования мест в гостиницах и пр.)

Информационно-решающие системы осуществляют, кроме того, операции переработки информации по определенному алгоритму. По **характеру использования выходной информации** такие системы принято делить на управляющие и советующие.

Результирующая информация управляющих ИС непосредственно трансформируется в принимаемые человеком решения. Для этих систем характерны задачи расчетного характера и обработка больших объемов данных. (Например, ИС планирования производства или заказов, бухгалтерского учета.)

Советующие ИС вырабатывают информацию, которая принимается человеком к сведению и учитывается при формировании управленческих решений, а не инициирует конкретные действия. Эти системы имитируют интеллектуальные процессы обработки знаний, а не данных. (Например, экспертные системы.)

В зависимости от **сферы применения** различают следующие классы ИС.

Информационные системы организационного управления - предназначены для автоматизации функций управленческого персонала как промышленных предприятий, так и непромышленных объектов (гостиниц, магазинов и пр.).

Основными функциями подобных систем являются: оперативный контроль и регулирование, оперативный учет и анализ, перспективное и оперативное планирование, бухгалтерский учет, управление сбытом, снабжением и другие экономические и организационные задачи.

ИС управления технологическими процессами (ТП) - служат для автоматизации функций производственного персонала по контролю и управлению производственными операциями. В таких системах обычно

предусматривается наличие развитых средств измерения параметров технологических процессов (температуры, давления, химического состава и т.п.), процедур контроля допустимости значений параметров и регулирования технологических процессов.

ИС автоматизированного проектирования (САПР) - предназначены для автоматизации функций инженеров-проектировщиков, конструкторов, архитекторов, дизайнеров при создании новой техники или технологии. Основными функциями подобных систем являются: инженерные расчеты, создание графической документации (чертежей, схем, планов), создание проектной документации, моделирование проектируемых объектов.

Интегрированные (корпоративные) ИС - используются для автоматизации всех функций фирмы и охватывают весь цикл работ от планирования деятельности до сбыта продукции. Они включают в себя ряд модулей (подсистем), работающих в едином информационном пространстве и выполняющих функции поддержки соответствующих направлений деятельности.

1.3. Классификация БД и СУБД

Классификация - разделение множества на подмножества по неформально предложенному признаку. В силу многогранности баз данных и СУБД (комплекса технических и программных средств, для хранения, поиска, защиты и использования данных) имеется множество классификационных признаков. Классификация БД по основным признакам приведена на рис. 2.1.

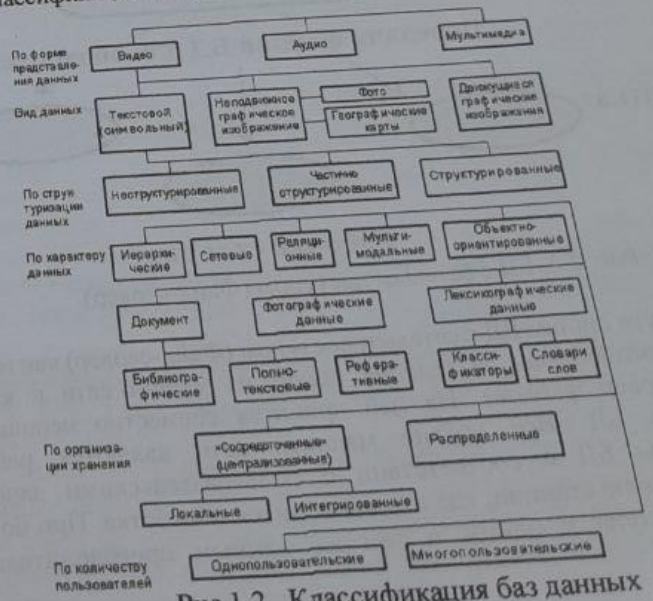


Рис 1.2. Классификация баз данных

Базы данных могут классифицироваться и с точки зрения экономической: по условиям предоставления услуг - бесплатные и платные (бесприбыльные, коммерческие); по форме собственности - государственные, негосударственные; по степени доступности - общедоступные, с ограниченным кругом пользователей.

В мире существует множество СУБД. Несмотря на их различие, все они опираются на единый устоявшийся комплекс основных понятий.

СУБД носит централизованный характер. Что предполагает необходимость существования некоторого лица (группы лиц), на которое возлагаются функции администрирования данными, хранимыми в базе.

По технологии обработки данных БД делятся на централизованные БД и распределённые БД.

Централизованная БД хранится в памяти одной вычислительной системы (применяется в локальных сетях ПК).

Централизованные БД могут быть с сетевым доступом.

Архитектуры систем централизованных БД с сетевым доступом подразделяются на файл-сервер и клиент-сервер.

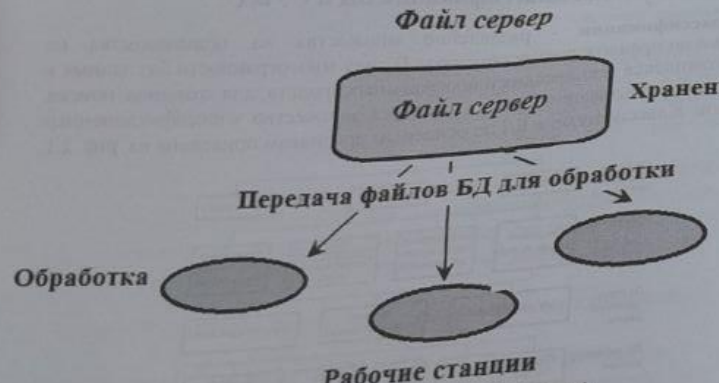


Рис. 2.2. БД с сетевым доступом (Файл-сервер)

Архитектура систем БД с сетевым доступом (Файл-сервер) как показано на рис. 2.2 предполагает выделение одной из машин сети в качестве центральной (сервер файлов). На ней хранится совместно используемая централизованная БД. Все другие машины сети являются рабочими станциями. Файлы БД в соответствии с пользовательскими запросами передаются на рабочие станции, где и производится обработка. При большой интенсивности доступа к одним и тем же данным производительность системы падает.

Рис. 2.3. БД с сетевым доступом Клиент – сервер

В архитектуре Клиент-сервер (рис. 2.3) подразумевается, что помимо хранения централизованной БД центральная машина (сервер базы данных) должна обеспечивать выполнение основного объёма обработки данных. Запрос на данные клиента, порождает поиск и извлечение данных на сервере. Извлечённые данные (но не файлы) транспортируются по сети от сервера к клиенту.

Пример БД - деловой ежедневник, в котором каждому календарному дню выделено по странице. Даже в отсутствии там записей, он не перестаёт быть ежедневником, т.к. имеет структуру, отличающую его от записных книжек, рабочих тетрадей и т.п. Другие примеры БД: база данных больных в поликлинике, БД по видеофильмам (видеотека), БД по сотрудникам организации (Ф.И.О., пол, дата рождения, место жительство, телефон, состав семьи и т.д.).

Распределённая БД состоит из нескольких частей, хранимых в различных ЭВМ вычислительной сети (работа с такой БД происходит с помощью СУБД).

По способу доступа к данным БД разделяются на БД с локальным и удаленным доступом.

БД с **локальным доступом** называется, если эта вычислительная система является компонентом сети ЭВМ, возможен распределённый доступ к такой базе. Такой способ использования БД часто применяют в локальных сетях ПК.

БД с **удалённым (сетевым) доступом** называется когда, части БД мог пересекаться или даже дублироваться, но хранятся в различных ЭВ вычислительной сети.

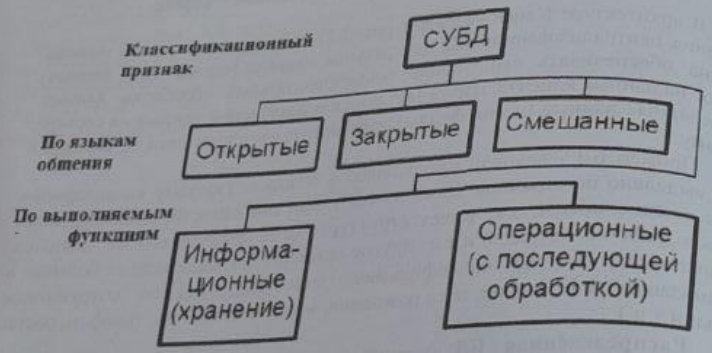
Для работы с созданной БД пользователю или администратору следует иметь перечень файлов-таблиц с описанием состава их да (структуры, схемы). Для этого создается специальный файл, называ словарем данных (депозитарием, словарем-справочником, энциклопе Описание БД относится к метаинформации.

В качестве технических средств могут выступать супер- или персональные компьютеры с соответствующими периферийными устройствами.

1.4. Классификация СУБД

Система управления базами данных (СУБД) - это совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями.

Системы управления базами данных следует классифицировать отдельно (рис. 2.4).



1.5. Определение понятия базы данных

База данных – это совместно используемый интегрированный набор логически связанных данных, хранящихся вместе с описанием этих данных, предназначенные для удовлетворения информационных потребностей организации.

ПОЯСНЕНИЕ:

Совместно используемый означает, что данными из базы данных ользуются многие пользователи с помощью различных средств доступа.

Интегрированный означает, что вместо разрозненных файлов с быточными данными здесь все данные собраны вместе с минимальной ей избыточности.

Логическая связанность данных в базе данных означает, что в ней ится информация о взаимосвязанных объектах (сущностях) реального , причем связи между объектами также хранятся в базе данных.

Набор понятий реального мира, который моделируется в базе данных, называют **предметной областью**.

Описание данных также хранится в базе данных. В совокупности описание данных называется **словарем данных** , а сами элементы описания – **метаданными** (данными о данных). Наличие метаданных позволяет достичь определенной независимости между программами и данными.

База данных – совокупность взаимосвязанных данных конкретной предметной области различного назначения.

1.6. Требования к базам данных

Итак, хорошо спроектированная база данных:

- Удовлетворяет всем требованиям пользователей к содержимому базы данных. Перед проектированием базы необходимо провести обширные исследования требований пользователей к функционированию базы данных.
- Гарантирует непротиворечивость и целостность данных. При проектировании таблиц нужно определить их атрибуты и некоторые правила, ограничивающие возможность ввода пользователем неверных значений. Для верификации данных перед непосредственной записью их в таблицу база данных должна осуществлять вызов правил модели данных и тем самым гарантировать сохранение целостности информации.
- Обеспечивает естественное, легкое для восприятия структурирование информации. Качественное построение базы позволяет делать запросы к базе более "прозрачными" и легкими для понимания; следовательно, снижается вероятность внесения некорректных данных и улучшается качество сопровождения базы.
- Удовлетворяет требованиям пользователей к производительности базы данных. При больших объемах информации вопросы сохранения производительности начинают играть главную роль, сразу "высвечивая" все недочеты этапа проектирования.

1.7. Определение понятия банка данных

Современной формой организации хранения и доступа к информации является банк данных.

Банк данных – система, специальным образом организованных данных (база данных), программных, технических, языковых средств, предназначенных для обеспечения централизованного накопления и коллективного многоцелевого использования данных.

Из определения следует, что банк данных – это сложная система, содержащая одну или несколько баз данных и все обеспечивающ подсистемы (инфраструктуру), необходимые для функционирования систе автоматизированной обработки данных.

1.8. Преимущества банковской организации данных

- возможность обеспечить не противоречивость и целостность информации
- возможность обращения к этой информации при решении различных задач
- сокращение избыточности хранимых данных и естественно затрат на создание и хранение данных и поддержание их в актуальном состоянии
- сокращение документа оборота, времени обмена информации в организации
- централизованное управление данными со стороны администратора базы данных
- возможность реализации принципа независимости прикладных программ от данных
- возможность работы с базой данных не только профессионалов , но и любых категорий пользователей

1.9. Требования к банку данных

- адекватность в отображении предметной области (полнота, целостность и непротиворечивость данных , актуальность информации)
- возможность взаимодействия пользователей различных категорий и в различных режимах
- обеспечение высокой эффективности доступа к данным
- дружественность интерфейса и малое время на освоение системы
- обеспечение секретности и конфиденциальности для некоторой части данных, определение полномочий различных групп пользователей
- обеспечение взаимной независимости программ и данных
- обеспечение надежности функционирования банка данных (защита данных от случайного и преднамеренного разрушения, возможность быстрого и полного восстановления данных в случае их разрушения)
- технологичность обработки данных
- приемлемые характеристики функционирования банка данных (стоимость обработки, время реакции системы на запросы, требуемые им машинные ресурсы)

1.10. Классификация банка данных

Большинство классификационных признаков относятся к базе данных, как к главной компоненте банка данных.

- по форме представления информации различают видеосистемы, аудиосистемы и системы мультимедиа
- по организации данных базы данных могут быть структурированные, не структурированные, частично структурированные (тексты, гипертексты). Структурированные базы данных по типу используемой модели делят на иерархические, сетевые, реляционные, смешанные и мультимодельные
- по типу хранимой информации различают документальные (библиографические, реферативные), фактографические и лексикографические (словари) базы данных
- по характеру организации хранения данных базы данных могут быть локальными (персональными), распределенными (по сети).

Контрольные вопросы

1. Что такое информация?
2. Что такое данные?
3. Инфологический и даталогический аспекты понятия «информация»?
4. Развитие автоматизированных информационных систем (АИС).
5. Требовании к БнД
6. Что такое база данных?
7. Что такое банк данных?
8. Преимущества банковской организации данных
9. Требования к банку данных
10. Классификация банка данных
11. Классификация информационных систем
12. На какие классы разделяется информационные системы по степени автоматизации и их описание.
13. На какие классы разделяется информационные системы **по типу хранимых данных** и их описание.
14. На какие классы разделяется информационные системы **по характеру обработки данных** и их описание.

Тестовые задания

1. Что такое предметная область

*A) часть реального мира, которая описывается или моделируется помощью базы данных и используемых их приложений

B) Совокупность данных объектах реального мира

C) Данные, хранящиеся в базе данных, как отображение части реальн мира

D) Набор информационных объектов

2. Что понимается под информацией?

*A) любые сведения о каком-либо событии, сущности, процессе и т.п
B) Извести средств массовой информации
C) Информации которых человек может увидит
D) Данные на диске

3. Какие вопросы рассматриваются в **инфологическом аспекте** данных?
A) вопросы представления данных в памяти
B) вопросы представления данных в памяти и связанные со смысловым содержанием данных
C) вопросы связанные с преобразованием данных
*D) вопросы, связанные со смысловым содержанием данных

4. Какие вопросы рассматриваются в **датологическом аспекте** данных?
*A) вопросы представления данных в памяти
B) вопросы представления данных в памяти и связанные со смысловым содержанием данных
C) вопросы связанные с преобразованием данных
D) вопросы, связанные со смысловым содержанием данных

5. **Банк данных** (БнД) — это…..
A) это область применения конкретного БД
B) это файлы - содержащие данных
*C) это АИС, включающая в свой состав комплекс специальных методов и средств (математических, информационных, программных, языковых, организационных и технических) для поддержания динамической информационной модели предметной области с целью обеспечения информационных запросов пользователей
D) это совместно используемый интегрированный набор логически связанных данных, хранящихся вместе с описанием этих данных, предназначенные для удовлетворения информационных потребностей организации

6. Банк данных (БнД) — это …
A) этот понятие эквивалентно к понятию базы данных
B) это техническое средство информационной системы
*C) АИС, включающий в свой состав комплекс специальных методов и средств
D) это распределенная база данных находящихся в разных хостах глобальной сети

7. **Что такое база данных (БД)** – это совместно используемый интегрированный набор логически связанных данных, хранящихся вместе с описанием этих данных, предназначенные для удовлетворения информационных потребностей организации.
A) набор файлов сохраняющие разнородные информации о предметной области

B) комплекс программных средств для обработки данных
*C) **База данных** – это совместно используемый интегрированный набор логически связанных данных, хранящихся вместе с описанием этих данных, предназначенные для удовлетворения информационных потребностей организации.
D) это телекоммуникационная система предназначенные для обмена информациями внутри организации

8. Перечислите требований к базам данных
*A) Удовлетворяет всем требованиям пользователей к содержимому базы данных.
*B) Гарантирует непротиворечивость и целостность данных.
*C) Обеспечивает естественное, легкое для восприятия структурирование информации.
*B) Удовлетворяет требованиям пользователей к производительности базы данных.

9. На какие классы делятся Информационные Системы по типу хранимых данных?
*A) фактографические и документальные
B) ручные и автоматические
C) автоматические и автоматизированные
D) информационно-поисковые и информационно-решающие

10. На какие классы делятся Информационные Системы основываясь на степени автоматизации информационных процессов?
A) ручные и автоматические
B) автоматические и автоматизированные
C) фактографические и документальные
*D) ручные, автоматические и автоматизированные

11. На какие классы делятся Информационные Системы в зависимости от **характера обработки** данных?
A) управляющие и советующие
*B) информационно-поисковые и информационно-решающие
C) ручные, автоматические и автоматизированные
D) информационно-поисковые и ручные

12. В зависимости от **сферы применения** различают следующие классы Информационных Систем.
A) Информационные системы организационного управления
B) управления технологическими процессами
*C) все правильно
D) ИС автоматизированного проектирования и Интегрированны (корпоративные) ИС

13. По **характеру использования выходной информа** информационные системы принято делить на…..
*A) управляющие и советующие

B) информационно-поисковые и информационно-решающие
C) ручные, автоматические и автоматизированные
D) информационно-поисковые и ручные

2. ТРЕХУРОВНЕВАЯ АРХИТЕКТУРА БАЗЫ ДАННЫХ

2.1. Уровни организации БД

Американским комитетом по стандартизацииANSI (American National Standards Institute) предложена трехуровневая система организации БД.

1. Уровень внешних моделей— самый верхний уровень, где каждая модель имеет свое «видение» данных. Этот уровень определяет точку зрения на БД отдельных приложений. Каждое приложение видит и обрабатывает только те данные, которые необходимы именно этому приложению. Например, система распределения работ использует сведения о квалификации сотрудника, но ее не интересуют сведения об окладе, домашнем адресе и телефоне сотрудника, и наоборот, именно эти сведения используются в подсистеме отдела кадров.

2. Концептуальный уровень— центральное управляющее звено, здесь база данных представлена в наиболее общем виде, который объединяет данные, используемые всеми приложениями, работающими с данной базой данных. Фактически концептуальный уровень отражает обобщенную модель предметной области (объектов реального мира), для которой создавалась база данных. Как любая модель, концептуальная модель отражает только существенные, с точки зрения обработки, особенности объектов реального мира.

3. Физический уровень— собственно данные, расположенные в файлах или в страничных структурах, расположенных на внешних носителях информации.

Эта архитектура позволяет обеспечить логическую (между уровнями 1 и 2) и физическую (между уровнями 2 и 3) независимость при работе с данными. Логическая независимость предполагает возможность изменения одного приложения без корректировки других приложений, работающих с этой же базой данных. Физическая независимость предполагает возможность переноса хранимой информации с одних носителей на другие при сохранении работоспособности всех приложений, работающих с данной базой данных. Это именно то, чего не хватало при использовании файловых систем.

2.2. Трехуровневая архитектура базы данных

СУБД должна предоставлять доступ к данным любым пользователям, включая и тех, которые практически не имеют и (или) не хотят иметь представления о:

- физическом размещении в памяти данных и их описаний;
- механизмах поиска запрашиваемых данных;

• проблемах, возникающих при одновременном запросе одних и тех же данных многими пользователями (прикладными программами);
• способах обеспечения защиты данных от некорректных обновлений и (или) несанкционированного доступа;
• поддержании баз данных в актуальном состоянии
• и множестве других функций СУБД.
• При выполнении основных из этих функций СУБД должна использовать различные описания данных.

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

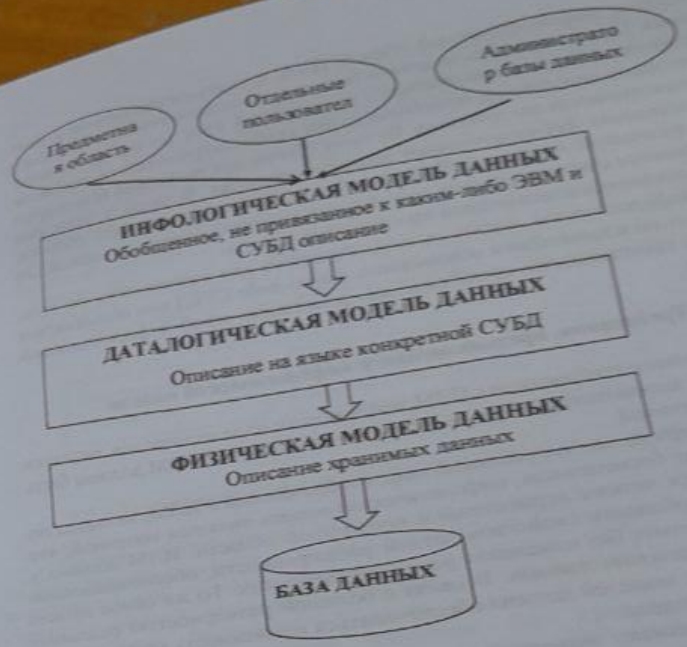
Объединяя частные представления о содержимом базы данных, полученные в результате опроса пользователей, и свои представления о данных, которые могут потребоваться в будущих приложениях, сначала создается обобщенное неформальное описание создаваемой базы данных. Это описание, выполненное с использованием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

Такая человеко-ориентированная модель полностью независима от физических параметров среды хранения данных. В конце концов этой средой может быть память человека, а не ЭВМ. Поэтому инфологическая модель не должна изменяться до тех пор, пока какие-то изменения в реальном мире не потребуют изменения в ней некоторого определения, чтобы эта модель продолжала отражать предметную область.

Остальные модели, показанные на рисунке, являются компьютеро-ориентированными. С их помощью СУБД дает возможность программам и пользователям осуществлять доступ к хранимым данным лишь по их именам, не заботясь о физическом расположении этих данных. Нужные данные отыскиваются СУБД на внешних запоминающих устройствах по ***физической модели данных.***

Так как указанный доступ осуществляется с помощью конкретной СУБД, то модели должны быть описаны на языке описания данных этой СУБД. Такое описание, создаваемое АБД по инфологической модели данных, называют ***даталогической моделью данных***.

Трехуровневая архитектура (инфологический, даталогический и физический уровни) позволяет обеспечить *независимость хранимых данных* от использующих их программ. Можно при необходимости переписать хранимые данные на другие носители информации и (или) реорганизовать их физическую структуру, изменив лишь физическую модель данных. Можно подключить к системе любое число новых пользователей (новых приложений), дополнив, если надо, даталогическую модель.



Указанные изменения физической и даталогической моделей не будут замечены существующими пользователями системы (окажутся "прозрачными" для них), так же как не будут замечены и новые пользователи. Следовательно, независимость данных обеспечивает возможность развития системы баз данных без разрушения существующих приложений.

2.3. Инфологическое моделирование

Общие сведения об инфологическом моделировании

В базе данных отображается какая-то часть реального мира. Естественно, что полнота ее описания будет зависеть от целей создаваемой информационной системы.

Для того чтобы база данных адекватно отражала предметную област проектировщик базы данных должен хорошо представлять себе все нюан

присущие данной предметной области (ПО), и уметь отобразить их в базе данных. Предметная область должна быть предварительно описана. Для этого в принципе может использоваться и естественный язык, но его применение имеет много недостатков, основными из них являются громоздкость описания и неоднозначность его трактовки. Поэтому обычно для этих целей используют искусственные формализованные языковые средства. В связи с этим под инфологической моделью(ИЛМ) понимают описание предметной области, выполненное с использованием специальных языковых средств, не зависящих от используемых в дальнейшем программных средств.

Инфологическая модель должна строиться вне зависимости от того, будете ли вы в дальнейшем использовать какую-либо СУБД или пользоваться другими программными средствами для реализации своей информационной системы.

Требования, предъявляемые к инфологической модели

Основным требованием к ИЛМ, вытекающим из ее назначения, является требование адекватного отображения предметной области. ИЛМ должна быть непротиворечивой.

Несмотря на то, что реальный мир, отображаемый в ИЛМ, является по своей природе бесконечным, инфологическая модель является конечной, что обеспечивается четким ограничением предметной области. ИЛМ должна в связи с этим обладать свойством легкой расширяемости, обеспечивающим ввод новых данных без изменения ранее определенных. То же самое можно сказать и об удалении данных. В связи с большой размерностью реальных инфологических моделей должна обеспечиваться возможность композиции и декомпозиции модели.

Инфологическая модель должна легко восприниматься разными категориями пользователей. Желательно, чтобы ИЛМ строил специалист, работающий в этой предметной области, а не проектировщик систем машинной обработки данных или хотя бы проверить сделанное описание, чтобы убедиться, что специфика предметной области воспринята правильно. Инфологическая модель должна также легко и однозначно восприниматься всеми специалистами, которые в дальнейшем участвуют в процессе проектирования баз данных и программного обеспечения.

Она является ядром системы проектирования. ИЛМ содержит необходимую и достаточную информацию для дальнейшего проектирования автоматизированной системы обработки информации.

Компоненты инфологической модели

Инфологическая модель предметной области включает в себя ряд компонентов (рис. 1). Центральной компонентой мифологической модели

является описание объектов предметной области и связей между ними(ER-модель)

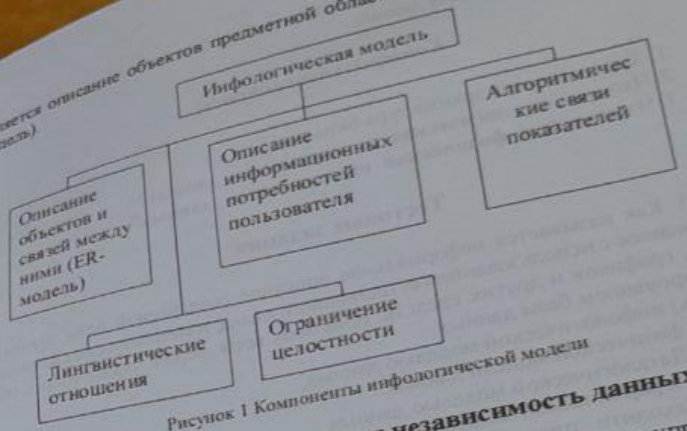


Рисунок 1 Компоненты инфологической модели

2.4. Логическая и физическая независимость данных

По мере накопления опыта использования первых систем управления базами данных довольно скоро стало очевидным, что необходим дополнительный уровень независимости данных. Общая логическая структура данных, как правило, сложная, и по мере роста базы данных она неизбежно изменяется. Поэтому важно обеспечить возможность изменения общей логической структуры без изменения при этом использующих ее многочисленных прикладных программ. В некоторых системах изменение общей логической структуры данных составляет форму ее существования, т. е. эта структура находится в состоянии постоянного развития. Поэтому требуются два уровня независимости данных. Мы будем называть их логической и физическойнезависимостью данных.

Логическая независимость данных означает, что общая логическая структура данных может быть изменена без изменения прикладных программ (изменение, конечно, не должно заключаться в удалении из базы данных таких элементов, которые используются прикладными программами).

Физическая независимость данных означает, что физическое расположение и организация данных могут изменяться, не вызывая при этом изменений ни общей логической структуры данных, ни прикладных программ.

Программное обеспечение баз данных будет фактически получать представление данных прикладного программиста из общей логической

структуры, а затем будет отображать общую логическую структуру а физическое представление данных.

Контрольные вопросы

1. Трехуровневая архитектура базы данных
2. Что означает **логическая независимость данных?**
3. Что означает **физическая независимость данных?**

Тестовые задания

1. Как называется неформальное описание создаваемой базы данных, выполненное с использованием естественного языка, математических формул, таблиц, графиков и других средств, понятных всем людям, работающих над проектированием базы данных.

*A) инфологической моделью данных
B) физической модели данных
C) Даталогической моделью данных
D) Иерархической моделью данных

2. Находите правильное выказывание о Инфологической модели данных.

A) зависит от вычислительных систем предприятия
B) зависит от программного обеспечения предприятия
C) Зависит от операционной платформы компьютеров предприятия
*D) полностью независимо от физических параметров среды хранения данных

3. Который из этих моделей данных является человеко-ориентированным?

A) Даталогическая модель данных
B) физическая модель данных
*C) инфологическая модель данных
D) Иерархическая модель данных

4. Который из этих моделей данных не является компьютерно-ориентированным?

A) Даталогическая модель данных
B) физическая модель данных
*C) инфологическая модель данных
D) Иерархическая модель данных

5. Как называется модель данных описанное на языке описания данных конкретной СУБД по инфологической модели данных.

A) инфологической моделью данных
B) физической модели данных

*C) Даталогической моделью данных
D) Иерархической моделью данных

6. Что означает логическая независимость данных?

A) Независимость логических структур данных от операционных систем
B) Независимость логических структур данных от сетевой технологии
*C) Независимость логических структур данных от прикладных программ
B) Независимость логических структур данных от компьютера

7. Физическая независимость данных означает....

A) Независимость физической организация данных от операционных систем
B) Независимость физической организация данных от сетевой организации данных
*C) Независимость физической организация данных от логической структуры данных
B) Независимость физической организация данных от компьютера

3. МОДЕЛИ БАЗЫ ДАННЫХ. МОДЕЛЬ СУЩНОСТЬ-СВЯЗЬ

3.1. Обзор ранних (дореляционных) СУБД

Рассмотрим некоторые наиболее общие характеристики ранних систем.

1. Эти системы активно использовались в течение многих лет, дольше, чем используется многие из реляционных СУБД. На самом деле некоторые из ранних систем используются даже в наше время, накоплены громадные базы данных, и одной из актуальных проблем информационных систем является использование этих систем совместно с современными системами.

2. Все ранние системы не основывались на каких-либо абстрактных моделях. Понятие модели данных фактически вошло в обиход специалистов в области БД только вместе с реляционным подходом. Абстрактные представления ранних систем появились позже на основе анализа и выявления общих признаков у различных конкретных систем.

3. В ранних системах доступ к БД производился на уровне записей. Пользователи этих систем осуществляли явную навигацию в БД, используя языки программирования, расширенные функциями СУБД. Интерактивный доступ к БД поддерживался только путем создания соответствующих прикладных программ с собственным интерфейсом.

4. Навигационная природа ранних систем и доступ к данным на уровне записей заставляли пользователя самого производить всю оптимизацию доступа к БД, без какой-либо поддержки системы.

5. После появления реляционных систем большинство ранних систем было оснащено "реляционными" интерфейсами. Однако в большинстве случаев это не сделало их по-настоящему реляционными системами, поскольку оставалась возможность манипулировать данными в естественном для них режиме.

3.2. Системы, основанные на инвертированных списках

К числу наиболее известных и типичных представителей таких систем относятся Datacom/DB компанииApplied Data Research, Inc. (ADR), ориентированная на использование на машинах основного класса фирмы IBM, и Adabas компании Software AG.

Организация доступа к данным на основе инвертированных списков используется практически во всех современных реляционных СУБД, но в этих системах пользователи не имеют непосредственного доступа к инвертированным спискам(индексам).

3.2.1. Структуры данных

В базе данных, организованной с помощью инвертированных списков хранимые таблицы и пути доступа к ним видны пользователям. При этом:

1. Строки таблиц упорядочены системой в некоторой физической последовательности.
2. Физическая упорядоченность строк всех таблиц может определяться и для всей БД (так делается, например, в Datacom/DB).
3. Для каждой таблицы можно определить произвольное число ключей поиска, для которых строятся индексы. Эти индексы автоматически поддерживаются системой, но явно видны пользователям.

3.2.2. Манипулирование данными

Поддерживаются два класса операторов:

3. Операторы, устанавливающие адрес записи, среди которых:

- прямые поисковые операторы(например, найти первую запись таблицы по некоторому пути доступа);
- операторы, находящие запись в терминах относительной позиции от предыдущей записи по некоторому пути доступа.

3. Операторы над адресуемыми записями

Типичный набор операторов:

1. Найти первую запись таблицы Т в физическом порядке;
2. Найти первую запись таблицы Т с заданным значением ключа поиска K;
3. Найти запись, следующую за записью с заданным адресом в заданном пути доступа;
4. Найти следующую запись таблицы Т в порядке пути поиска с заданным значением K;
5. Найти первую запись таблицы Т в порядке ключа поиска K со значением ключевого поля, большим заданного значения K;
6. Выбрать запись с указанным адресом;
7. Обновить запись с указанным адресом;
8. Удалить запись с указанным адресом;
9. Включить запись в указанную таблицу; операция генерирует адрес записи.

3.3. Иерархическая модель

Типичным представителем (наиболее известным и распространенным) является Information Management System (IMS) фирмыIBM. Первая версия появилась в1968 г. До сих пор поддерживается много баз данных, что создает существенные проблемы с переходом как на новую технологию БД, так и на новую технику.

3.1. Иерархические структуры данных

Иерархическая БД состоит из упорядоченного набора деревьев; более точно, из упорядоченного набора нескольких экземпляров одного типа дерева. Тип дерева (рис.1) состоит из одного"корневого" типа записи и упорядоченного набора из нуля или более типов поддеревьев(каждое из которых является некоторым типом дерева). Тип дерева в целом представляет собой иерархически организованный набор типов записи.

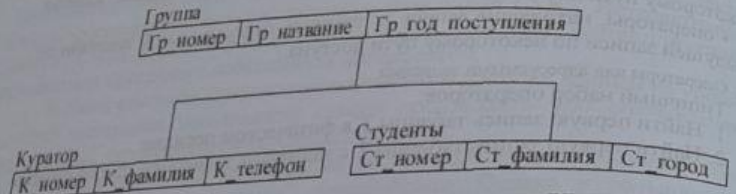


Рисунок 1 Пример типа дерева (схемы иерархической БД)

Здесь (рис. 1) Группа является предком для Куратора и Студенты, а Куратор и Студенты– потомки Группа. Между типами записи поддерживаются связи. База данных с такой схемой могла бы выглядеть следующим образом (рис. 2).

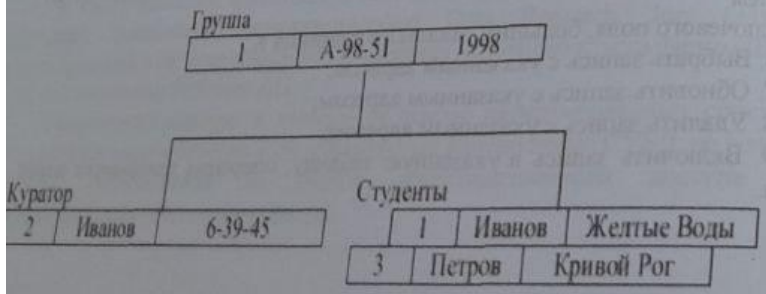


Рисунок 2. Один экземпляр дерева.

Все экземпляры данного типа потомка с общим экземпляром типа предка называются близнецами. Для БД определен полный порядок обхода– сверху - вниз, слева - направо.

В IMS использовалась оригинальная и нестандартная терминология: "сегмент" вместо"запись", а под"записью БД" понималось все дерево сегментов.

3.2. Манипулирование данными

Примерами типичных операторов манипулирования иерархически организованными данными могут быть следующие:

1. Найти указанное дерево БД(например, отдел310);
2. Перейти от одного дерева к другому;
3. Перейти от одной записи к другой внутри дерева(например, от отдела- к первому сотруднику);
4. Перейти от одной записи к другой в порядке обхода иерархии;
5. Вставить новую запись в указанную позицию;
6. Удалить текущую запись.

3.3. Ограничения целостности

Автоматически поддерживается целостность ссылок между предками и потомками. Основное правило: никакой потомок не может существовать без своего родителя. Заметим, что аналогичное поддержание целостности по ссылкам между записями, не входящими в одну иерархию, не поддерживается(примером такой"внешней" ссылки может быть содержимое поля Каф_Номер в экземпляре типа записи Куратор).

В иерархических системах поддерживалась некоторая форма представлений БД на основе ограничения иерархии. Примером представления приведенной выше БД может быть иерархия, изображенная на рис. 3

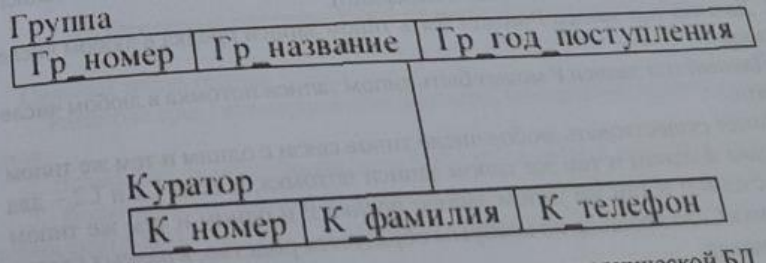


Рисунок 3. Пример представления иерархической БД.

3.4. Сетевая модель

Типичным представителем является Integrated Database Management System (IDMS) компании Cullinet Software, Inc., предназначенная для использования на машинах основного класса фирмы IBM под управлением большинства операционных систем. Архитектура системы основана на предложениях Data Base Task Group (DBTG) Комитета по языкам программирования Conference on Data Systems Languages (CODASYL), организации, ответственной за определение языка программирования Кобол. Отчет DBTG был опубликован в1971г., а в70-х годах появилось несколько систем, среди которых IDMS.

4.1 Сетевые структуры данных

Сетевой подход к организации данных является расширением иерархического. В иерархических структурах запись-потомок должна иметь в точности одного предка; в сетевой структуре данных потомок может иметь любое число предков. Сетевая БД состоит из набора экземпляров каждого типа записи и набора экземпляров каждого типа связи (рис. 4).

Тип связи определяется для двух типов записи: предка и потомка. Экземпляр типа связи состоит из одного экземпляра типа записи предка и упорядоченного набора экземпляров типа записи потомка. Для данного типа связи L с типом записи предка P и типом записи потомка C должны выполняться следующие два условия:

1. Каждый экземпляр типа P является предком только в одном экземпляре L;
2. Каждый экземпляр C является потомком не более, чем в одном экземпляре L.

На формирование типов связи не накладываются особые ограничения; возможны, например, следующие ситуации:

1. Тип записи потомка в одном типе связи L1 может быть типом записи предка в другом типе связиL2 (как в иерархии).
2. Данный тип записи P может быть типом записи предка в любом числе типов связи.
3. Данный тип записи P может быть типом записи потомка в любом числе типов связи.
4. Может существовать любое число типов связи с одним и тем же типом записи предка и одним и тем же типом записи потомка; и если L1 и L2 – два типа связи с одним и тем же типом записи предка P и одним и тем же типом записи потомка C, то правила, по которым образуется родство, в разных связях могут различаться.
5. Типы записи X и Y могут быть предком и потомком в одной связи и потомком и предком– в другой.

6. Предок и потомок могут быть одного типа записи.

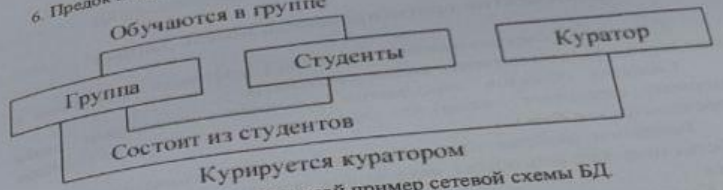


Рисунок 4. Простой пример сетевой схемы БД

4.2. Манипулирование данными

Примерный набор операций может быть следующим:

1. Найти конкретную запись в наборе однотипных записей (инженера Сидорова);
2. Перейти от предка к первому потомку по некоторой связи (к первому сотруднику отдела310);
3. Перейти к следующему потомку в некоторой связи (от Сидорова к Иванову);
4. Перейти от потомка к предку по некоторой связи (найти отдел Сидорова);
5. Создать новую запись;
6. Уничтожить запись;
7. Модифицировать запись;
8. Включить в связь;
9. Исключить из связи;
10. Переставить в другую связь и т.д.

4.5 Основные достоинства и недостатки ранних СУБД

Сильные места ранних СУБД:

1. Развитые средства управления данными во внешней памяти на низком уровне;
2. Возможность построения вручную эффективных прикладных систем;
3. Возможность экономии памяти за счет разделения подобъектов (в сетевых системах).

Недостатки:

1. Слишком сложно пользоваться;
2. Фактически необходимы знания о физической организации;

3. Прикладные системы зависят от этой организации;
4. Их логика перегружена деталями организации доступа к БД.

3.5. Объектно-ориентированная модель данных

Структура объектно-ориентированной БД графически представима в виде **дерева, узлами которого являются объекты**.

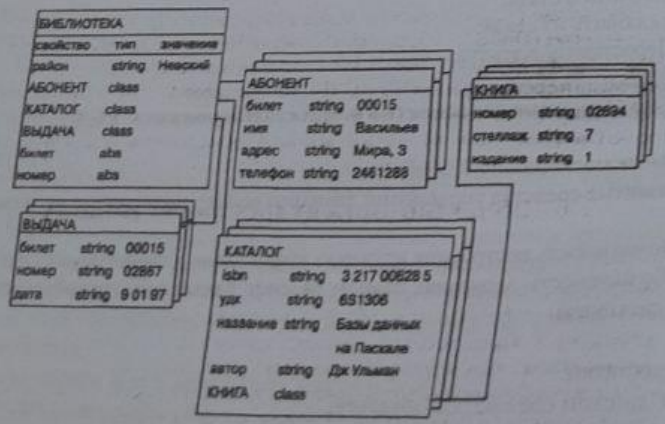
Свойства объектов описываются некоторым стандартным типом (например, строковым —string) или типом, конструируемым пользователем (определяется как **class**).

Значением свойства типа string является строка символов. Значение свойства типа class есть объект, являющийся экземпляром соответствующего класса.

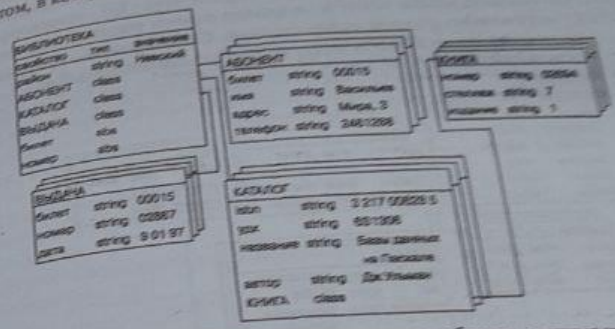
Каждый объект-экземпляр класса считается потомком объекта, в котором он определен как свойство. **Объект-экземпляр класса принадлежит своему классу и имеет одного родителя. Родовые отношения в БД образуют связную иерархию объектов.**

Здесь объект типа **БИБЛИОТЕКА** является родительским для объектов-экземпляров классов **АБОНЕНТ, КАТАЛОГИ, ВЫДАЧА**. Различные объекты типа **КНИГА** могут иметь одного или разных родителей. Объекты типа **КНИГА**, имеющие одного и того же родителя, должны различаться по крайней мере инвентарным номером (уникален для каждого экземпляра книги), но имеют одинаковые значения свойств *isbn, удк, название* и *автор*.

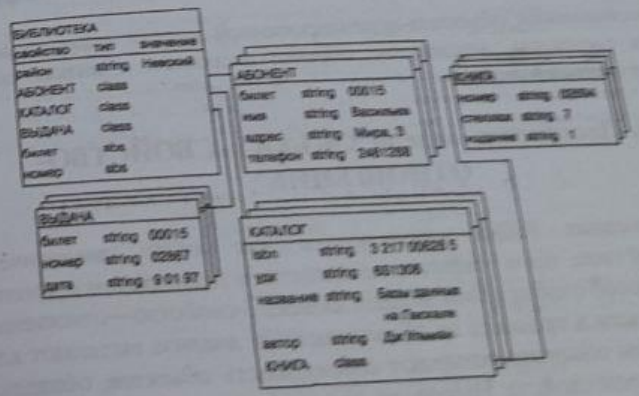
Логическая структура объектно-ориентированной БД внешне похожа на структуру **иерархической БД**. Основное отличие между ними состоит в методах манипулирования данными.



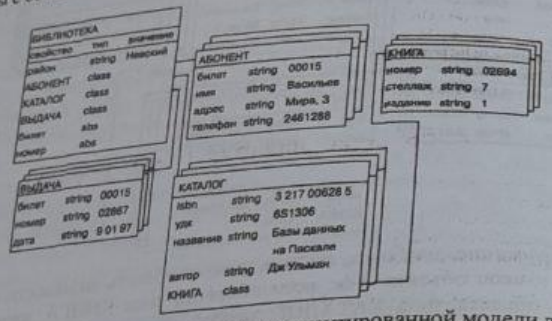
Инкапсуляция ограничивает *область видимости* имени свойства пределами того объекта, в котором оно определено. Так, если в объект типа КАТАЛОГ добавить свойство, задающее телефон автора книги и имеющее название *телефон*, то мы получим одноименные свойства у объектов АБОНЕНТ и КАТАЛОГ. Смысл такого свойства будет определяться тем объектом, в который оно инкапсулировано.



Наследование, наоборот, распространяет область видимости свойства на всех потомков объекта. Так, всем объектам типа КНИГА, являющимся потомками объекта типа КАТАЛОГ, можно приписать свойства объекта-родителя: *isbn, удк, название* и *автор*. Если необходимо расширить действие механизма наследования на объекты, не являющиеся непосредственными родственниками (например, между двумя потомками одного родителя), то в их общем предке определяется абстрактное свойство типа *abs*.



Полиморфизм в объектно-ориентированных языках программирования означает способность одного и того же программного кода работать с разнотипными данными. Другими словами, он означает допустимость в объектах разных типов иметь методы (процедуры или функции) с одинаковыми именами. Во время выполнения объектной программы одни и те же методы оперируют с разными объектами в зависимости от типа аргумента. Применительно к нашей объектно-ориентированной БД полиморфизм означает, что объекты класса КНИГА, имеющие разных родителей из класса КАТАЛОГ, могут иметь разный набор свойств. Следовательно, программы работы с объектами класса КНИГА могут содержать полиморфный код.



Основным ***достоинством*** объектно-ориентированной модели данных в сравнении с реляционной является возможность отображения информации о сложных взаимосвязях объектов.

Объектно-ориентированная модель данных позволяет **идентифицировать отдельную запись базы данных и определять функции их обработки.**

Недостатками объектно-ориентированной модели являются высокая понятийная сложность, неудобство обработки данных и низкая скорость выполнения запросов.

3.6. Построение модели“ОБЪЕКТ–СВОЙСТВО–ОТНОШЕНИЕ”

Для описания ИЛМ используются как языки аналитического (описательного) типа, так и графические средства в дальнейшем применяется графический способ отображения модели“объект—свойство—отношение”. В предметной области в процессе ее обследования и анализа выделяют классы объектов. Классом объектов называют совокупность объектов, обладающих одинаковым набором свойств. Например, если в качестве предметной области

рассмотреть вуз, то в ней можно выделить следующие классы объектов: учащиеся, преподаватели, аудитории и т. д. Объекты могут быть реальными, так и названные выше, а могут быть и абстрактными, как, например, предметы, которые изучают студенты.

При отражении в информационной системе каждый объект представляется своим идентификатором, который отличает один объект класса от другого, а каждый класс объектов представляется именем этого класса. Так, для каждого объекта класса“ИЗУЧАЕМЫЕ ПРЕДМЕТЫ” идентификатором должен быть уникальным. Идентификатор объекта будет“НАЗВАНИЕ ПРЕДМЕТА”.

Каждый объект обладает определенным набором свойств. Для объектов одного класса набор этих свойств одинаков, а их значения, естественно, могут различаться. Например, для объектов класса“СТУДЕНТ” таким набором свойств, описывающим объекты класса, может быть“ГОД РОЖДЕНИЯ”, “ПОЛ” и др.

При описании предметной области надо изобразить каждый из существующих классов объектов и набор свойств, фиксируемый для объектов данного класса.

Будем использовать для отображения объектов и их свойств следующие обозначения (рис. 2).

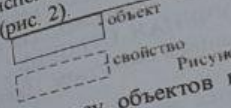


Рисунок 2 Обозначение объектов и их свойств

Каждому классу объектов в инфологической модели присваивается уникальное имя.

При построении инфологической модели желательно дать словесную интерпретацию каждой сущности, особенно если возможно неоднозначное толкование понятия.

Рисунок 3 Изображение связи «объект - свойство»

При описании предметной области надо отразить связи между объектом и характеризующими его свойствами. Это изображается просто в виде линии, соединяющей обозначение объекта и его свойств.

Связь между объектом и его свойством может быть различной. Объект может обладать только одним значением какого-то свойства. Например, каждый человек может иметь только одну дату рождения. Назовем такие свойства единичными. Для других свойств возможно существование одновременно нескольких значений у одного объекта. Пусть, например, при описании "СОТРУДНИКА" фиксируется в качестве его свойства "ИНОСТРАННЫЙ ЯЗЫК", которым он владеет. Так как сотрудник может знать несколько иностранных языков, то такое свойство будем называть множественным. При изображении связи между объектом и его свойствами для единичных свойств будем использовать одинарную стрелку, а для множественных свойств— двойную.

Кроме того, некоторые свойства являются постоянными, их значение не может измениться с течением времени. Назовем такие свойства статическими, а те свойства, значение которых может изменяться со временем, будем называть динамическими.

Другой характеристикой связи между объектом и его свойством является признак того, присутствует ли это свойство у всех объектов данного класса либо отсутствует у некоторых объектов. Например, для отдельных служащих может иметь место свойство "УЧЕНАЯ СТЕПЕНЬ", а другие объекты этого класса могут не обладать, указанным свойством. Назовем такие свойства условными.

В инфологической модели отображаются не отдельные экземпляры объектов, а классы объектов. Когда в ИЛМ изображено обозначение объекта, то ясно, что речь идет о классе объектов, обладающих описанными свойствами. Поэтому в инфологическую модель в большинстве случаев можно в явном виде не вводить еще и обозначение для класса объектов. Явное изображение класса объектов необходимо только в том случае, если в ПО для данного класса объектов фиксируются не только характеристики, относящиеся к отдельным объектам этого класса, но и какие-то интегральные характеристики, относящиеся ко всему классу в целом. Например, если для класса объектов "СОТРУДНИКИ ПРЕДПРИЯТИЯ" фиксируется не только возраст каждого из сотрудников, но и средний возраст всех сотрудников, то в инфологической модели необходимо отразить не только объект "СОТРУДНИК", но и класс объектов "СОТРУДНИКИ". Для отображения класса объектов можно использовать какое-то специфическое обозначение или такое же, которое используется для объектов.

3.7. Семантическое моделирование данных, ER-диаграммы

Широкое распространение реляционных СУБД и их использование в самых разнообразных приложениях показывает, что реляционная модель

данных достаточна для моделирования предметных областей. Однако проектирование реляционной базы данных в терминах отношений на основе кратко рассмотренного нами механизма нормализации часто представляет собой очень сложный и неудобный для проектировщика процесс.

При этом проявляется ограниченность реляционной модели данных в следующих аспектах:

• Модель не предоставляет достаточных средств для представления смысла данных. Семантика реальной предметной области должна независимым от модели способом представляться в голове проектировщика. В частности, это относится к упоминавшейся нами проблеме представления ограничений целостности.

• Для многих приложений трудно моделировать предметную область на основе плоских таблиц. В ряде случаев на самой начальной стадии проектирования проектировщику приходится производить насилие над собой, чтобы описать предметную область в виде одной (возможно, даже ненормализованной) таблицы.

• Хотя весь процесс проектирования происходит на основе учета зависимостей, реляционная модель не предоставляет каких-либо средств для представления этих зависимостей.

• Несмотря на то, что процесс проектирования начинается с выделения некоторых существенных для приложения объектов предметной области("сущностей") и выявления связей между этими сущностями, реляционная модель данных не предлагает какого-либо аппарата для разделения сущностей и связей.

На использовании разновидностей ER-модели основано большинство современных подходов к проектированию баз данных (главным образом, реляционных). Модель была предложена Ченом (Chen) в1976 г. Моделирование предметной области базируется на использовании графических диаграмм, включающих небольшое число разнородных компонентов. В связи с наглядностью представления концептуальных схем баз данных ER-модели получили широкое распространение в системах CASE, поддерживающих автоматизированное проектирование реляционных баз данных. Среди множества разновидностейER-моделей одна из наиболее развитых применяется в системе CASE фирмы ORACLE. Ее мы и рассмотрим. Более точно, мы сосредоточимся на структурной части этой модели.

Основными понятиями ER-модели являются сущность, связь и атрибут.

Сущность– это реальный или представляемый объект, информация о котором должна сохраняться и быть доступна. В диаграммах ER-модели сущность представляется в виде прямоугольника, содержащего имя сущности. При этом имя сущности- это имя типа, а не некоторого конкретного экземпляра этого типа. Для большей выразительности и лучшего понимания имя сущности может сопровождаться примерами конкретных объектов этого типа.

Ниже изображена сущность АЭРОПОРТ с примерными объектами Шереметьево и Хитроу:

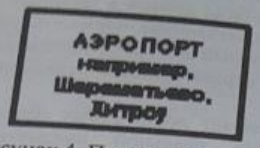


Рисунок 4. Пример сущности.

Каждый экземпляр сущности должен быть отличим от любого другого экземпляра той же сущности(это требование в некотором роде аналогично требованию отсутствия кортежей-дубликатов в реляционных таблицах).

Связь– это графически изображаемая ассоциация, устанавливаемая между двумя сущностями. Эта ассоциация всегда является бинарной и может существовать между двумя разными сущностями или между сущностью и ей же самой(рекурсивная связь). В любой связи выделяются два конца(в соответствии с существующей парой связываемых сущностей), на каждом из которых указывается имя конца связи, степень конца связи (сколько экземпляров данной сущности связывается), обязательность связи(т.е. любой ли экземпляр данной сущности должен участвовать в данной связи).

Связь представляется в виде линии, связывающей две сущности или ведущей от сущности к ней же самой. При это в месте"стыковки" связи с сущностью используются трехточечный вход в прямоугольник сущности, если для этой сущности в связи могут использоваться много(many) экземпляров сущности, и одноточечный вход, если в связи может участвовать только один экземпляр сущности. Обязательный конец связи изображается сплошной линией, а необязательный- прерывистой линией.

Как и сущность, связь– это типовое понятие, все экземпляры обеих пар связываемых сущностей подчиняются правилам связывания.

В изображенном ниже примере связь между сущностями БИЛЕТ и ПАССАЖИР связывает билеты и пассажиров. При том конец сущности с именем"для" позволяет связывать с одним пассажиром более одного билета, причем каждый билет должен быть связан с каким-либо пассажиром. Конец сущности с именем"имеет" означает, что каждый билет может принадлежать только одному пассажиру, причем пассажир не обязан иметь хотя бы один билет.

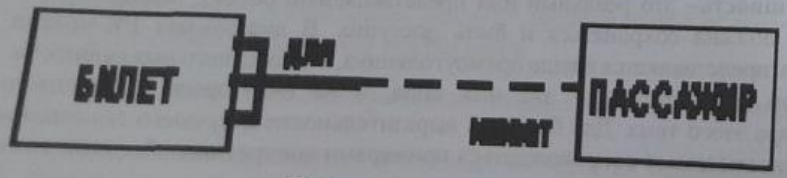


Рисунок 5. пример связи

Лаконичной устной трактовкой изображенной диаграммы
следующая:

• Каждый БИЛЕТ предназначен для одного и только одного ПАССАЖИРА;

• Каждый ПАССАЖИР может иметь один или более БИЛЕТОВ.

На следующем примере изображена рекурсивная связь, связывающая сущность ЧЕЛОВЕК с ней же самой. Конец связи с именем"сын" определяет тот факт, что у одного отца может быть более чем один сын. Конец связи с именем"отец" означает, что не у каждого человека могут быть сыновья.

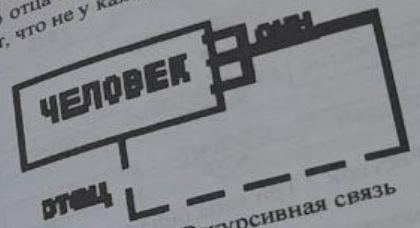


Рисунок 6. Рекурсивная связь

Лаконичной устной трактовкой изображенной диаграммы является следующая:

• Каждый ЧЕЛОВЕК является сыном одного и только одного ЧЕЛОВЕКА;

• Каждый ЧЕЛОВЕК может являться отцом для одного или более ЛЮДЕЙ ("ЧЕЛОВЕКОВ").

Атрибутом сущности является любая деталь, которая служит для уточнения, идентификации, классификации, числовой характеристики или выражения состояния сущности. Имена атрибутов заносятся в прямоугольник, изображающий сущность, под именем сущности и изображаются малыми буквами, возможно, с примерами.

3.8. Три типа бинарных связей

На рис. 7 показаны три типа бинарных связей. В связи1:1 («один одному»)

Одиночный экземпляр сущности одного типа связан с одиночным экземпляром сущности другого типа. На рис. 7, а с СЛУЖЕБНЫЙ_АВТОМОБИЛЬ связывает одиночную сущность из СОТРУДНИК с одиночной сущностью класса АВТОМОБИЛЬ соответствии с этой диаграммой, ни за одним сотрудником не закре более одного автомобиля, и ни один автомобиль не закреплен более одним сотрудником.

Рисунок 7 Три типа бинарных связей.

На рис. 7, *б* изображен второй тип связи, 1:N («один кN» или«один ко многим»), В этой связи, которая называется ОБЩЕЖИТИЕ-ЖИЛЕЦ, единичный экземпляр сущности класса ОБЩЕЖИТИЕ связан со многими экземплярами сущности класса СТУДЕНТ. В соответствии с этим рисунком, в общежитии проживает много студентов, но каждый студент живет только в одном общежитии.

Позиция, в которой стоят1 иN, имеет значение. Единица стоит на той стороне связи, где располагается ОБЩЕЖИТИЕ, аN стоит на той стороне связи, где располагается СТУДЕНТ. Если бы1 иN располагались наоборот, и связь записывалась бы какN:l, получилось бы, что в общежитии живет один студент, причем каждый студент живет в нескольких общежитиях. Это, разумеется, не так. На рис.7, в показан третий тип бинарной связи, N:M (читается«N к M» или«многие ко многим»). Эта связь называется СТУДЕНТ-КЛУБ, и она связывает экземпляры сущностей класса СТУДЕНТ с экземплярами сущностей класса КЛУБ. Один студент может быть членом нескольких клубов, а в одном клубе может состоять много студентов.

Числа внутри ромба, символизирующего связь, обозначают максимальное количество сущностей на каждой стороне связи. Эти ограничения называются максимальными кардинальными числами, а совокупность из двух таких ограничений для обеих сторон связи называется максимальной кардинальностью(maximum cardinality) связи. Например, о связи, изображенной на рис. 3.3, б, говорят, что она обладает максимальной кардинальностью1:N. Кардинальные числа могут иметь и другие значения, а только1 иN. Например, связь между сущностями БАСКЕТБОЛЬНАЯ

КОМАНДА и ИГРОК может иметь кардинальность1:5, что говорит нам о том, что в баскетбольной команде может быть не более пяти игроков.

Связи трех типов, представленных на рис. 7, называются иногда связями типа «ИМЕЕТ», или связями обладания(HAS-A relationships). Такой термин используется потому, что одна сущность имеет(has) связь с другой сущностью. Например: сотрудник имеет автомобиль, студент имеет общежитие, клуб имеет студентов.

Более сложные элементы ER-модели

Мы остановились только на самых основных и наиболее очевидных понятиях ER-модели данных. К числу более сложных элементов модели относятся следующие:

• Подтипы и супертипы сущностей. Как в языках программирования с развитыми типовыми системами(например, в языках объектно-ориентированного программирования), вводится возможность наследования типа сущности, исходя из одного или нескольких супертипов. Интересные нюансы связаны с необходимостью графического изображения этого механизма.

• Связи"many-to-many". Иногда бывает необходимо связывать сущности таким образом, что с обоих концов связи могут присутствовать несколько экземпляров сущности(например, все члены кооператива сообща владеют имуществом кооператива). Для этого вводится разновидность связи"многие-со-многими".

• Уточняемые степени связи. Иногда бывает полезно определить возможное количество экземпляров сущности, участвующих в данной связи(например, служащему разрешается участвовать не более, чем в трех проектах одновременно). Для выражения этого семантического ограничения разрешается указывать на конце связи ее максимальную или обязательную степень.

• Каскадные удаления экземпляров сущностей. Некоторые связи бывают настолько сильными(конечно, в случае связи"один-ко-многим"), что при удалении опорного экземпляра сущности(соответствующего концу связи"один") нужно удалить и все экземпляры сущности, соответствующие концу связи"многие". Соответствующее требование"каскадного удаления" можно сформулировать при определении сущности.

• Домены. Как и в случае реляционной модели данных бывает полезна возможность определения потенциально допустимого множества значений атрибута сущности (домена).

Диаграммы сущность-связь

Схемы, изображенные на рис. 3.3, называются диаграммами«сущность—связь», илиER-диаграммами(entity-relationship

diagrams, ER-diagrams). Такие диаграммы стандартизированы, но не слишком жестко. В соответствии с этим стандартом, классы сущностей обозначаются прямоугольниками, связи обозначаются ромбами, а максимальное кардинальное число каждой связи указывается внутри ромба1. Имя сущности указывается внутри прямоугольника, а имя связи указывается рядом с ромбом.

Рисунок8 Пример диаграммы«сущность-связь»

Контрольные вопросы

1. Основные конструктивные элементы инфологических моделей
2. Что такое сущность
3. Что такое тип сущности и экземпляр сущности
4. Что такое атрибут сущности
5. Что такое связь сущности
6. Типы связи сущностей

4. ПЛАНИРОВАНИЕ, ПРОЕКТИРОВАНИЕ И АДМИНИСТРИРОВАНИЕ БАЗЫ ДАННЫХ ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЙ БАЗ ДАННЫХ

База данных является фундаментальным компонентом информационной системы, а ее разработку и использование следует рассматривать с точки зрения самых широких требований организации. Следовательно, жизненный цикл информационной системы организации неотъемлемым образом связан с жизненным циклом системы базы данных, поддерживающей ее функционирование.

Жизненный цикл информационной системы обычно состоит из нескольких этапов: планирование, сбор и анализ требований, проектирование, создание прототипа, реализация, тестирование, преобразование данных и сопровождение.

Проектирование базы данных - процесс разработки структуры (схемы) базы данных (БД) в соответствии с требованиями пользователей и одна из наиболее сложных и ответственных задач, связанных с созданием информационной системы (ИС).

4.1. Требования к проекту базы данных

Основная цель процесса проектирования БД состоит в получении такого проекта, который удовлетворяет следующим требованиям:

Основные требования, которым должен удовлетворят проект базы данных (БД):

- 1. Корректность схемы БД.
- 2. Обеспечение ограничений на ресурсы вычислительной системы.
- 3. Эффективность функционирования.
- 4. Обеспечение защиты данных.
- 5. Гибкость.
- 6. Простота и удобство эксплуатации.

Удовлетворение первых 4-х требований обязательно для принятия проекта.

4.2. Жизненный цикл проектирование БД

Основные действия, выполняемые на каждом этапе жизненного цикл приложения базы данных.

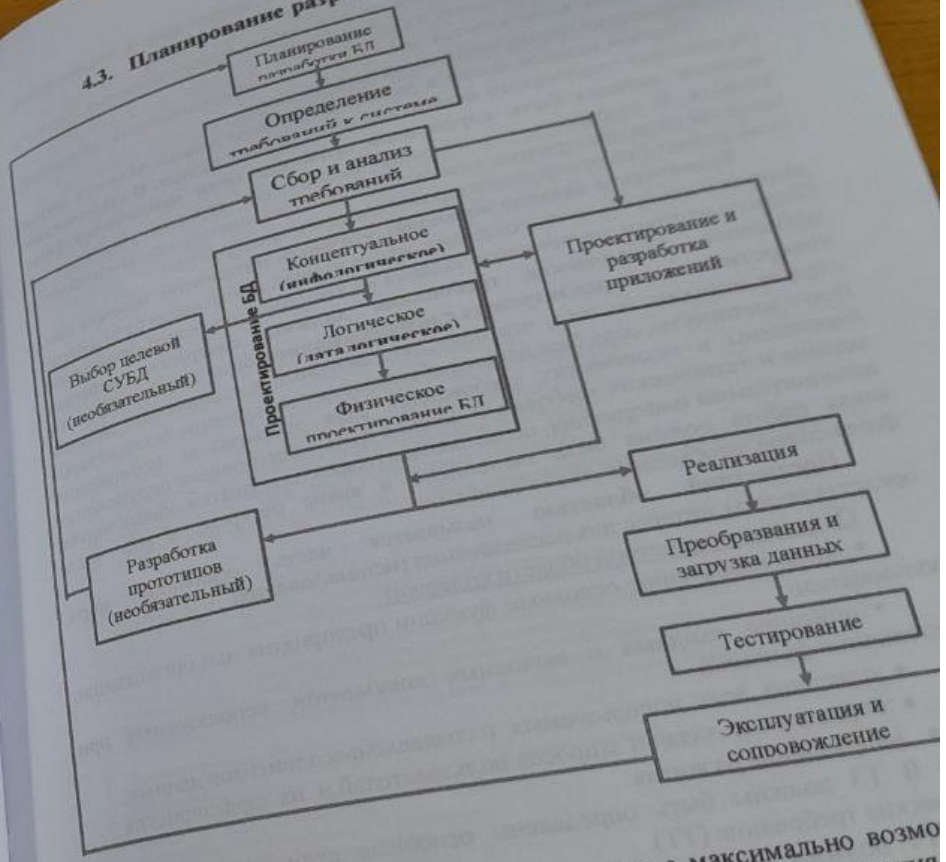
Этап	Описание
Планирование разработки базы данных	Планирование наиболее эффективного способа реализации этапов жизненного цикла системы
Определение требований к системе	Определение диапазона действий и границ приложения базы данных, состава его пользователей и областей применения
Сбор и анализ требований пользователей	Сбор и анализ требований пользователей из всех возможны* областей применения
Проектирование базы данных	Полный цикл разработки включает концептуальное, логическое и физическое проектирование базы данных
Выбор целевой СУБД (необязательный этап)	Выбор наиболее подходящей СУБД для приложения базы данных
Разработка приложений	Определение пользовательского интерфейса и прикладных программ, которые используют и обрабатывают данные в базе данных
Создание прототипов (необязательный этап)	Создание рабочей модели приложения базы данных, которая позволяет разработчикам или пользователям представить и оценить окончательный вид и способы функционирования системы
Реализация	Создание внешнего, концептуального и внутреннего определений базы данных и прикладных программ
Преобразование и загрузка данных	Преобразование и загрузка данных (и прикладных программ) из старой системы в новую
Тестирование	Приложение базы данных тестируется с целью обнаружения ошибок, а также его проверки на соответствие всем требованиям, выдвинутым пользователями
Эксплуатация и сопровождение	На этом этапе приложение базы данных считается полностью разработанным и реализованным. Впредь вся система будет находиться под постоянным наблюдением и соответствующим образом поддерживаться, В случае необходимости в функционирующее приложение могут вноситься изменения, отвечающие новым требованиям. Реализация этих изменений проводится посредством повторного выполнения некоторых из перечисленных выше этапов жизненного цикла

Специалисты, необходимые для выполнения этой работы:

- Аналитики (специалисты исследуемой предметной области).
- Пользователи – те работники, для которых создаётся АИС.
- Проектировщики (разработчики базы данных).
- Администраторы (системные, базы данных, безопасности и др.)
- Программисты (разработчики программного обеспечения).

Рис 2. Жизненный цикл проектирование БД. Этапы жизненного цикла

4.3. Планирование разработки базы данных



Подготовительные действия, позволяющие с максимально возмо
эффективностью реализовать этапы жизненного цикла приложения
данных.

Планирование разработки базы данных должно быть нера
связано с общей стратегией построения информационной
организации. При выработке такой стратегии необходимо решить сл
основные задачи:

• определение бизнес-планов и целей организации с последующим выделением ее потребностей в информационных технологиях;

• оценка показателей уже существующих информационных систем с целью выявления их сильных и слабых сторон;

• оценка возможностей использования информационных технологий для достижения преимуществ перед конкурентами.

Первым важным шагом в планировании базы данных является четкое определение технического задания для проекта базы данных. **В техническом задании должны быть определены основные цели приложения базы данных.** В разработке технического задания, как правило, участвуют те представители предприятия, которые стали инициаторами разработки проекта базы данных (например, директор или владелец предприятия).

Техническое задание позволяет уточнить назначение проекта базы данных и наметить пути к созданию эффективного приложения базы данных. После подготовки технического задания необходимо определить технические требования. **Технические требования должны содержать перечень конкретных задач, реализуемых с использованием базы данных.** При этом следует исходить из того, что цели, поставленные в техническом задании, будут достигнуты, если база данных обеспечивает выполнение задач, которые определены в технических требованиях. Для обоснования технического задания и технических требований должна быть подготовлена определенная дополнительная информация, позволяющая охарактеризовать в общих чертах, какая работа должна быть выполнена и какие ресурсы, в том числе финансовые, необходимо на это выделить.

Предметной областью называется часть реального мира, представляющая интерес для исследования (использования).

Описание предметной области содержит:

• цель, назначение, основные функции предприятия или организации, пользователи;

• описание входных и выходных документов, используемых при выполнении функций;

• описание всех используемых и создаваемых элементов данных;

• определение задач и запросов пользователей и их характеристик;

• направление развития.

• В ТЗ должны быть определены основные цели приложения БД, технические требования (ТТ).

• ТТ должны содержать перечень конкретных задач, реализуемых с использованием БД.

• В разработке ТЗ участвуют инициаторы разработки проекта БД (директор или владелец предприятия).

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

- **"Каковы задачи вашей компании?"**
- **"Для чего, по вашему мнению, необходимо создать базу данных?"**
- **"Почему вы думаете, что база данных поможет решить ваши проблемы?"**

Фрагмент технического задания для приложения базы данных DreamHome

"Приложение базы данных DreamHome предназначено для обработки данных, используемых при оформлении сделок по аренде недвижимости между клиентами и владельцами недвижимости, а также для обеспечения совместного доступа к информации из всех отделений компании".

ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

- "Каковы ваши должностные обязанности?"
- "Какого вида задачи вы повседневно выполняете?"
- "С данными какого рода вы обычно работаете?"
- "Какого типа отчеты вы обычно используете?"
- "Дела какого типа вам необходимо отслеживать?"
- "Какие услуги предоставляет ваша компания своим заказчикам?"

Обработка (ввод, модификация и удаление) данных по отделениям.
Обработка (ввод, модификация и удаление) данных по персоналу.
Обработка (ввод, модификация и удаление) данных по объектам недвижимости, сдаваемым в аренду.
Обработка (ввод, модификация и удаление) данных по владельцам недвижимости.
Обработка (ввод, модификация и удаление) данных по клиентам.
Обработка (ввод, модификация и удаление) данных по осмотру недвижимости.
Обработка (ввод, модификация и удаление) данных по договорам об аренде.
Обработка (ввод, модификация и удаление) данных по объявлениям в газетах.
Обработка (ввод, модификация и удаление) данных по отделениям.

Выполнение поиска по отделениям.
Выполнение поиска по персоналу.
Выполнение поиска по объектам недвижимости, сдаваемым в аренду.
Выполнение поиска по владельцам недвижимости.
Выполнение поиска по клиентам.
Выполнение поиска по осмотру недвижимости.
Выполнение поиска по договорам об аренде.
Выполнение поиска по объявлениям в газетах.

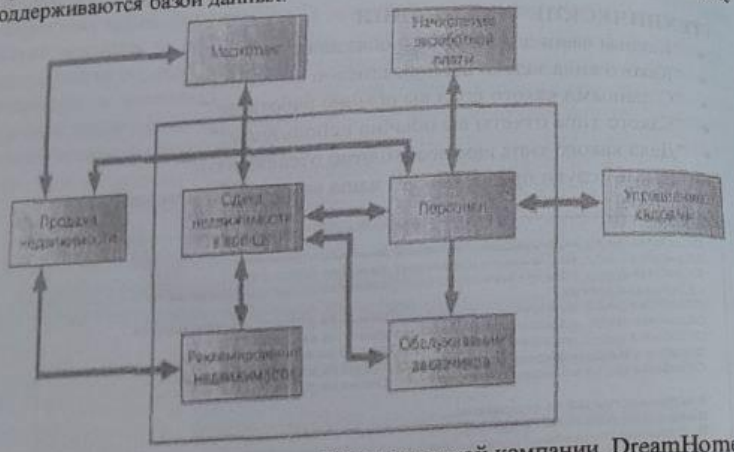
Контроль состояния дел по объектам недвижимости, сдаваемым в аренду.
Контроль состояния дел по клиентам, желающим арендовать недвижимость.
Контроль состояния дел по договорам об аренде.

Формирование отчетов по отделениям.
Формирование отчетов по персоналу.
Формирование отчетов по объектам недвижимости, сдаваемым в аренду.
Формирование отчетов по владельцам недвижимости.
Формирование отчетов по клиентам.
Формирование отчетов по осмотру недвижимости.
Формирование отчетов по договорам об аренде.
Формирование отчетов по объявлениям в газетах.

4.4. Определение требований к системе

Определение требований к системе. Определение диапазона действия и задач приложения базы данных, состава его пользователей и областей применения.

Прежде чем перейти к проектированию приложения базы данных, важно установить задачи исследуемой системы и способы взаимодействия приложения с другими частями информационной системы организации. Эти задачи должны учитывать не только работу текущих пользователей и области применения разрабатываемой системы, но и будущих пользователей и области возможные области применения. На следующем слайде приведена схема, которая определяет состав задач и область применения приложения базы данных учебного проекта риэлтерской компании *DreamHome*. Кроме описания области применения приложения базы данных, необходимо определить основные пользовательские представления, которые поддерживаются базой данных.



Задачи системы для приложения БД риэлтерской компании DreamHome

Пользовательское представление

• **Пользовательское** представление. Определение требований к приложению базы данных конкретной категории пользователей (таких как менеджер или инспектор) или потребностей подразделения предприятия (таких как отдел маркетинге, отдел кадров или отдел снабжения).В приложении базы данных может быть предусмотрено одно или несколько пользовательских представлений. Определение пользовательских представлений является существенной составляющей разработки приложения

базы данных, поскольку позволяет гарантировать, чтобы ни одна важная категория пользователей базы данных не была исключена из рассмотрения при разработке требований к новому приложению. Пользовательские представления являются особенно полезными при создании относительно сложных приложений базы данных, поскольку позволяют разделить всю совокупность требований к базе данных на легко анализируемые группы.Любое **пользовательское представление** определяет требования к приложению базы данных в части хранимых в ней данных и транзакций, выполняемых над данными (т.е. оно определяет, какие действия и над какими данными должен выполнять тот или иной пользователь). Требования пользовательского представления могут относиться только к данному представлению или частично совпадать с требованиями других представлений.

4.5. Сбор и анализ требований пользователей

Методики сбора фактов о предметной области

Сбор **и** анализ требований **пользователей.** Процесс сбора и анализа информации о той части организации, работа которой будет поддерживаться с помощью создаваемого приложения базы данных, а также использование этой информации для определения требований пользователей к создаваемой системе.

Существует много методов сбора этой информации, известных под общим названием методов сбора фактов:

- Изучение документации;
- Проведение собеседований;
- Наблюдение за работой предприятия;
- Проведение исследований;
- Проведение анкетирования.

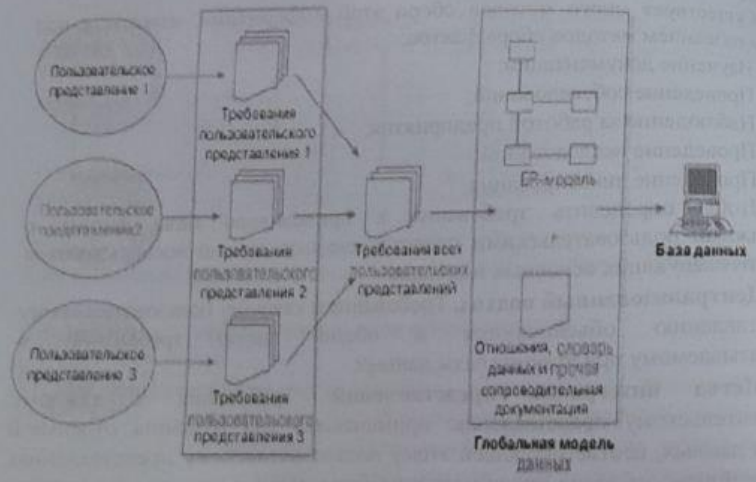
Чтобы определить требования к приложению базы данных с несколькими пользовательскими представлениями, можно воспользоваться одним из следующих основных подходов.

•**Централизованный подход.** Требования к каждому пользовательскому предоставлению объединяются в общий набор требований к разрабатываемому приложение базы данных.

•**Метод интеграции представлений**. Требования к каждому пользовательскому представлению применяются для создания отдельной модели данных, соответствующей этому пользовательскому представлению. В дальнейшем, на этапе проектирования базы данных полученные модели данных объединяются.

•**Сочетание обоих подходов.**

Пример применения централизованного подхода к управлению пользовательскими представлениями 1-3



Пример применения метода интеграции представлений к управлению пользовательскими представлениями

4.6. Процесс проектирования БД

•**Проектирование базы данных.** Процесс создания проекта базы данных, предназначенной для поддержки функционирования предприятия и способствующей достижению его целей.

• Подходы к проектированию базы данных. Существуют два основных подхода к проектированию систем баз данных: ***нисходящий и восходящий***. При **восходящем подходе** работа начинается с самого нижнего уровня атрибутов (т.е. свойств сущностей и связей), которые на основе анализа существующих между ними связей группируются в отношения, представляющие типы сущностей и связи между ними.

•**Восходящий подход** в наибольшей степени приемлем для проектирования простых баз данных с относительно небольшим количеством атрибутов.

•Более подходящей стратегией проектирования сложных баз данных является использование **нисходящего подхода**. Начинается этот подход с разработки моделей данных, которые содержат несколько высокоуровневых сущностей и связей, затем работа продолжается в виде серии нисходящих уточнений низкоуровневых сущностей, связей и относящихся к ним

атрибутов. Инсходящий подход демонстрируется в концепции модели "сущность-связь".

Процесс проектирования БД включает в себя следующие этапы:

- ***Концептуальное (инфологическое) проектирование.***

1. Определение требований к операционной обстановке, в которой будет функционировать информационная система.

2. Выбор системы управления базой данных (СУБД) и других инструментальных программных средств.

- ***Логическое (даталогическое) проектирование БД.***
- ***Физическое проектирование БД.***

Концептуальное (инфологическое) проектирование. Процесс создания модели используемой на предприятии информации, независящей от любых физических аспектов её представления. *Основными задачами инфологического проектирования* являются определение предметной области системы и формирование взгляда на ПО с позиций сообщества будущих пользователей БД, т.е. инфологической модели ПО.

Рассмотрим *основные подходы* к созданию инфологической модели предметной области

- **Функциональный подход к проектированию БД.** Этот метод реализует принцип "от задач" и применяется тогда, когда известны функции некоторой группы лиц и/или комплекса задач, для обслуживания информационных потребностей которых создаётся рассматриваемая БД.
- **Предметный подход к проектированию БД.** Предметный подход к проектированию БД применяется в тех случаях, когда у разработчиков есть чёткое представление о самой ПО и о том, какую именно информацию они хотели бы хранить в БД, а структура запросов не определена или определена не полностью.
- **Проектирование с использованием метода "сущность-связь"**

Логическое проектирование базы данных. Процесс создания модели используемой на предприятии информации на основе выбранной модели организации данных, но без учета типа целевой СУБД и других физических аспектов реализации.

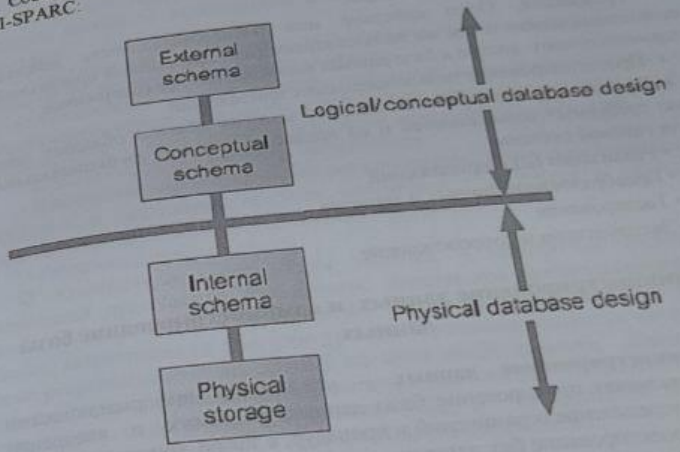
Концептуальная модель данных, созданная на предыдущем этапе, уточняется и преобразуется в логическую модель данных. Логическая модель данных учитывает особенности выбранной модели организации данных в **целевой СУБД** (например, реляционная модель).

Для проверки правильности логической модели данных используется ***метод нормализации***. Созданная логическая модель данных является источником информации для этапа физического проектирования.

Физическое проектирование базы данных. Процесс подготовки описания реализации базы данных на вторичных запоминающих устройствах; на этом этапе рассматриваются основные отношения, организация файлов и

индексов, предназначенных для обеспечения эффективного доступа к данным, а также все связанные с этим ограничения целостности и средства защиты.

Соответствие этапов моделирования данных и элементов архитектуры ANSI-SPARC:



4.7. Выбор СУБД и других программных средств

Этап процедуры выбора СУБД

1. Определение предметной области проводимого исследования
2. Сокращение списка выбора до двух-трех продуктов
3. Оценка продуктов
4. Проведение обоснованного выбора и подготовка отчета

Разработчики руководствуются критериями:

- тип модели данных, которую поддерживает данная СУБД, её адекватность потребностям рассматриваемой предметной области;
- характеристики производительности системы;
- запас функциональных возможностей для дальнейшего развития ИС;
- степень оснащённости системы инструментарием для персонала администрирования данными;
- удобство и надежность СУБД в эксплуатации;
- стоимость СУБД и дополнительного программного обеспечения.

4.8. Продолжение. Стадии жизненного цикла разработки системы с базой данных

- *Проектирование приложений*
- *– Проектирование транзакций*
- *Транзакция. Одно действие или последовательность действий, выполняемых одним и тем же пользователем (или прикладной программой), которые получают доступ к базе данных или изменяют ее содержимое.*
- *Прототипирование пользовательских приложений*
- *Прототип — это рабочая модель, которая обычно обладает лишь частью требуемых возможностей и не предоставляет всех функциональных средств готовой системы.*
- • Реализация БД и приложений
- • Преобразование данных и загрузка
- • Тестирование
- • Эксплуатация и сопровождение

4.9. Администрирование данных и администрирование базы данных

• **Администрирование данных** – управление информационными ресурсами, включая планирование базы данных, разработку и внедрение стандартов, определение ограничений и процедур, а также концептуальное и логическое проектирование баз данных.

• Решаемые задачи:

• Выбор подходящих инструментов разработки.

• Помощь в разработке корпоративных стратегий создания информационной системы, развития информационных технологий и бизнес-стратегий.

• Предварительная оценка осуществимости проектов и планирование процесса создания базы данных.

• Разработка корпоративной модели данных.

• Определение требований организации к используемым данным.

• Определение стандартов сбора данных и выбор формата их представления.

• Оценка объемов данных и вероятности их роста.

• Определение способов и интенсивности использования данных

• Концептуальное и логическое проектирование базы данных

Определение правил доступа к данным и мер безопасности, соответствующих правовым нормам и внутренним требованиям организации.

• Взаимодействие с АБД и разработчиками приложений с целью обеспечения соответствия создаваемых приложений всем существующим требованиям.

• Обучение пользователей — изучение существующих стандартов обработки данных и юридической ответственности за их некорректное применение.

• Постоянная модернизация используемых информационных систем и технологий по мере развития бизнес-процессов.

• Обеспечение полноты всей требуемой документации, включая корпоративную модель, стандарты, ограничения, процедуры, использование словаря данных, а также управление работой конечных пользователей.

• Поддержка словаря данных организации.

• Взаимодействие с конечными пользователями для определения новых требований и разрешения проблем, связанных с доступом к данным и недостаточной производительностью их обработки.

• Разработка правил защиты.

○ **Администрирование базы данных** – управление физической реализацией приложений баз данных: физическое проектирование базы данных и ее реализация, организация поддержки целостности и защиты данных, наблюдение за текущим уровнем производительности системы, а также реорганизация базы данных по мере необходимости. Деятельность администратора баз данных (АБД) является технической в большей мере, чем деятельность администратора данных (АД), и предусматривает знание особенностей конкретных СУБД и операционных систем.

Решаемые задачи:

• Оценка и выбор целевой СУБД.

• Физическое проектирование базы данных.

• Реализация физического проекта базы данных в среде целевой СУБД.

• Определение требований защиты и поддержки целостности данных.

Взаимодействие с разработчиками приложений баз данных.

• Разработка стратегии тестирования.

• Обучение пользователей.

• Ответственность за сдачу в эксплуатацию готового приложения базы данных.

• Контроль текущей производительности системы и соответствующая настройка базы данных.

• Регулярное резервное копирование.

• Разработка требуемых механизмов и процедур восстановлений.

• Обеспечение полноты используемой документации, включая материалы, разработанные внутри организации.

• Поддержка актуальности используемого программного и аппаратного обеспечения, включая заказ и установку пакетов обновлений в случае необходимости.

Сравнение задач администрирования данных и базы данных

Администратор данных	Администратор базы данных
Участвует в стратегическом планировании информационной системы организации	Оценивает новые СУБД
Определяет долгосрочные цели	Выполняет планы достижения целей
Применяет стандарты, правила и процедуры	Применяет стандарты, правила и процедуры
Определяет требования к данным	Реализует требования к данным
Выполняет концептуальное и логическое проектирование базы данных	Выполняет логическое и физическое проектирование базы данных
Разрабатывает и сопровождает корпоративную модель данных	Реализует физический проект базы данных
Координирует разработку системы	Выполняет текущий контроль и управление базой данных
Управленческая направленность	Техническая направленность
Работа АД не зависит от типа целевой СУБД	Работа АБД зависит от типа целевой СУБД

Контрольные вопросы

1. Опишите процесс проектирование БД.
7. Какие требования должно удовлетворяет проект базы данных?
8. Из каких этапов состоит жизненный цикл проектирование БД? Приведите схему этапов.
9. Коротко опишете каждую этап жизненного цикла проектирование БД.
10. Планирование разработки базы данных.
11. Как составляется техническое задание и техническое требования для проекта базы данных? Состав ТЗ и ТТ?
12. В чем состоит назначение пользовательских представлений в контексте приложения базы данных?
13. В чем состоит назначение сбора и анализа требований пользователей в контексте приложения базы данных?
14. Подходы сбора и анализа требований пользователей в контексте приложения базы данных?
15. Подходы к проектированию базы данных.

16. 3 этапа процесса проектирования БД.
17. Концептуальное (инфологическое) проектирование БД. Основные подходы к созданию инфологической модели предметной области
18. Логическое проектирование базы данных.
19. Физическое проектирование базы данных.
20. Назовите основные этапы выбора СУБД и опишите типичный подход к выбору "оптимальной" СУБД.
21. В чем состоят цели и задачи администрирования данных и администрирования базы данных?

5. РЕЛЯЦИОННАЯ МОДЕЛЬ ДАННЫХ. ВЫПОЛНЕНИЯ ОПЕРАЦИЙ В РЕЛЯЦИОННЫХ МОДЕЛЯХ. ОТНОШЕНИЯ И СВЯЗИ. РЕЛЯЦИОННАЯ АЛГЕБРА И ЭЛЕМЕНТЫ РЕЛЯЦИОННЫХ ВЫЧИСЛЕНИЙ

5.1. Реляционная модель данных

Реляционная модель данных предложена сотрудником фирмы IBM Эдгаром Коддом в 1970 г. Основная идея реляционной модели данных заключается в том, чтобы упростить структуру базы данных. В ней отсутствуют явные указатели на предков и потомков, а все данные представлены в виде простых таблиц, разбитых на строки и столбцы. Основным понятием в реляционной модели является «отношение» (relation).

В *реляционной* модели используется своеобразная терминология, но это не меняет сущности модели. Так, на логическом уровне элемент чаще всего называют атрибутом; кроме того, для него используются термины "колонка", "столбец", "поле". Совокупность атрибутов образует кортеж (ряд, запись, строку). Совокупность кортежей образует отношение (таблицу или файл БД).

Связи между файлами в реляционной модели в явном виде могут не описываться. Они устанавливаются динамически в момент обработки данных по равенству значений соответствующих полей. Структуры записей в реляционных БД – линейные.

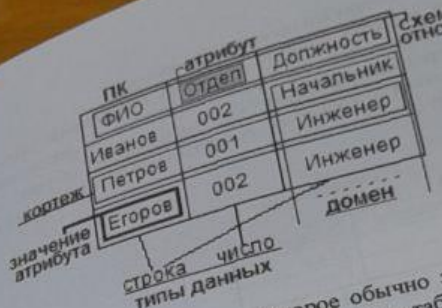
Каждое отношение по определению имеет ключ, т.е. атрибут (простой ключ) или совокупность атрибутов (составной ключ), однозначно идентифицирующий кортеж.

Атрибут или группа атрибутов, которая в рассматриваемом отношении не является ключом, а в другом отношении ключом является, называется *внешним* ключом.

Если какая-то таблица содержит внешний ключ, то она: а) логически связана с таблицей, содержащей соответствующий первичный ключ; б) эта связь имеет характер один ко многим.

Реляционная модель была разработана Коддом в 1969 – 70 гг. на основе математической теории отношений и опирается на систему понятий, важнейшими из которых являются таблица, отношение, строка, столбец, первичный ключ, внешний ключ.

Реляционной считается такая модель данных, в которой все данные представлены для пользователя в виде прямоугольных таблиц значений данных, и все операции над базой данных сводятся к манипулированию таблицами. Таблица состоит из строк и столбцов и имеет имя, уникальное внутри базы данных. Таблица отражает тип объекта реального мира (сущность), а каждая ее строка – конкретный объект. Рассмотрим основные понятия реляционных моделей на примере "Сотрудник" – сущность.



Каждый столбец имеет имя, которое обычно записывается в верхней части таблицы. Оно должно быть уникальным в таблице, однако различные таблицы могут иметь столбцы с одинаковыми именами.

Любая таблица должна иметь по крайней мере один столбец; столбцы расположены в таблице в соответствии с порядком следования их имен при ее создании. В отличие от столбцов (атрибутов), строки не имеют имен, порядок их следования не определен, а количество не ограничено.

Любая таблица имеет один или несколько столбцов, значения которых однозначно идентифицирует каждую ее строку. Первичный ключ в примере (рис. 4.7) – это столбец "Номер детали".

Значения атрибутов выбираются из наименьшей информационной единицы – домена. Другими словами, домен – это множество всех возможных значений атрибута объекта. Рассмотрим еще два понятия "Степень" и "Кардинальное число". Под кардинальным числом отношения понимают количество кортежей, а степень отношения – это количество атрибутов данного отношения.

Взаимосвязь таблиц является важнейшим элементом реляционной модели данных. Она поддерживается внешними ключами. Рассмотрим пример, в котором БД хранит информацию о сотрудниках (таблица "Сотрудник") и руководителях (таблица "Руководитель") в некоторой организации (рис. 4.8).

Первичный ключ таблицы "Руководитель" – столбец "Номер". Столбец "Фамилия" не является уникальным, поэтому не применяется в качестве первичного ключа. Столбец "Номер Руководителя" является внешним ключом в таблице "Сотрудник".

В БД дополнительно к самим данным должен храниться словарь данных и другие объекты, например, экранные формы, отчеты, просмотры (*views*) и прикладные программы.

Ограничения целостности в реляционных БД требуют, чтобы, например значения атрибутов выбирались только из соответствующего домена, и внешний ключ не может быть указателем на несуществующую строку таблице (целостность по ссылке).

Рассмотрим более подробно понятие "Отношение".

В реляционных моделях следует различать переменные отношения, таким значения отношений. Переменная отношения – это обычная переменная, такая же, как и в языках программирования, т.е. именованный объект, значением которого может изменяться со временем. В значение этой переменной в любой момент времени и будет значением отношения.

Сотрудник

№	Фамилия	№ руководителя	Должность
4781	Иванов	5742	Инж.
5125	Петров	6931	Зам.нач.
3120	Сидоров	5742	Слес.
1230	Лапин	2345	Инж.
2130	Юдин	6931	Нач.отд.

Внешний ключ

Руководитель

№	Фамилия	Отдел	Стаж
5742	Иванов	САПР	15
6931	Петров	№5	25
2345	Сидоров	Лаборатория	21

Первичный ключ

Рис. 4.8 Взаимосвязь таблиц

5.2. Введение в реляционную алгебру

Реляционная алгебра, определенная Коддом, состоит из восьми операторов, составляющих две группы.

В первую группу входят традиционные операции над множествами: объединение (U), пересечение (∩), вычитание (–) и декартово произведение (*). Все операции модифицированы с учетом того, что их операндами являются отношения, а не произвольные множества.

Вторую группу образуют специальные реляционные операции: выборка, проекция, соединение и деление.

Рассмотрим подробнее результаты этих операций над отношениями.

Объединение U. Возвращает отношение, содержащее все кортежи, которые принадлежат одному из двух определенных отношений или обоим (рис. 5.1, *а*).

Пересечение ∩. Возвращает отношение, содержащее все кортежи, которые принадлежат одновременно двум определенным отношениям (рис. 5.1, *б*).

Вычитание – Возвращает отношение, содержащее все кортежи, которые принадлежат первому из двух определенных отношений и не принадлежат второму (рис. 5.1, *в*).

Декартово произведение *. Возвращает отношение, содержащее всевозможные кортежи, которые являются сочетанием двух кортежей, принадлежащих соответственно двум определенным отношениям (рис. 5.1, *г*).

Выборка – возвращает отношение, содержащее все кортежи из определенного отношения, которое удовлетворяет определенным условиям. С точки зрения алгебраических операций это *ограничение* (рис. 5.2, *а*).

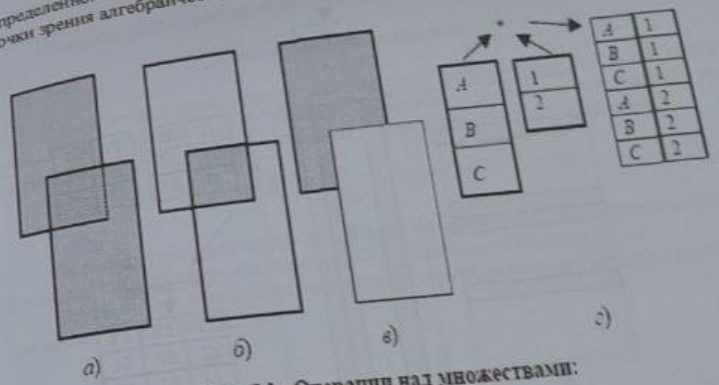


Рис. 5.1 Операции над множествами:
а – объединение; *б* – пересечение;
в – вычитание; *г* – декартово произведение

Проекция – возвращает отношение, содержащее все кортежи (подкортежи) определенного отношения после исключения из него некоторых атрибутов (рис. 5.2, *б*).

Соединение – возвращает отношение, кортежи которого – это сочетания двух кортежей (принадлежащих соответственно двум определенным), имеющих общее значение для одного или нескольких общих атрибутов этих двух отношений. Общие значения в результирующем кортеже появляются только один раз, а не дважды. Такое соединение называют естественным соединением (рис. 5.2, *в*).

Деление – для двух отношений бинарного и унарного, возвращает отношение, содержащее все значения одного атрибута бинарного отношения, которые соответствуют всем значениям в унарном отношении (рис. 5.3).

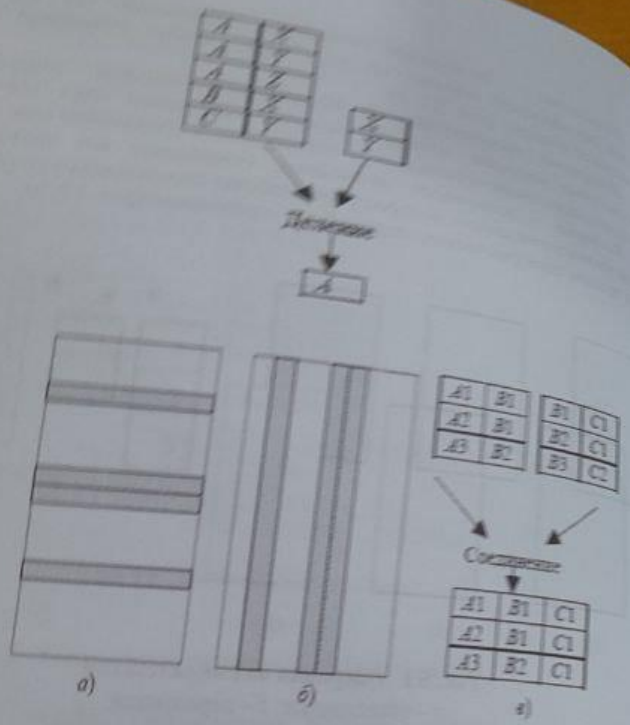


Рис. 5.2 *Специальное реляционное отношение:*
а – выборка, б – проекция, в – соединение

Результат каждой операции над отношением также является отношением. Это реляционное свойство называется *замкнутости*.

Результат одной операции может использоваться в качестве исходных данных для другой. Следова- тельно, существует возможность, например, взять проекцию от объединения, или соединение от двух выборок и т.д. Такие выражения считаются вложенными.

Каждое отношение включает заголовок, тело, множество потенциальных ключей. При выполнении реляционных операций необходимо рассмотреть набор правил наследования имен атрибутов и потенциальных ключей.

5.3. Стандартные реляционные операции

Рассмотрим выполнение операций над отношениями подробнее.

Для операций объединения (*union*), пересечения (*intersect*) и вычитания (*minus*) должны выполняться два свойства:

- операнды должны иметь одну и ту же степень;
- соответствующие атрибуты должны быть определены на одном и том же домене.

Операция умножения не требует выполнения этих условий.

Традиционные операции

Объединением двух совместимых по типу отношений A и B (A *union* B) называется отношение C с тем же заголовком и телом, состоящим из множества кортежей t_i, принадлежащих A или B или обоим отношениям.

$$C = (A \text{ union } B) \mid t_i \in C \; \forall \; t_j \in A \; \& \; t_i \in C \; \forall \; t_j \in B.$$

Пример: пусть отношения A и B будут такими, как они отражены ниже: A – детали изготовленные из стали; B – детали весом больше 0,5 кг.

Тогда A *union* B представляет детали, которые *или* изготовлены из стали, *или* имеют вес больше 0,5 кг.

A

К	Назва-ние детали	Вес	Мате-риал
K1	D1	0.8	Сталь
K2	D2	1.0	Сталь
K3	D3	0.5	Сталь

B

К	Назва-ние детали	Вес	Матери-ал
K1	D1	0.8	Сталь
K2	D2	1.0	Сталь
K4	D4	0.7	Алюми-ний

В результате получим 4 кортежа, а не 6 – повторяющиеся значения удаляются.

C

K	Название детали	Вес	Материал
K1	*D1*	0.8	Сталь
K2	*D2*	1.0	Сталь
K3	*D2*	0.5	Сталь
K4	*D4*	0.7	Алюминий

Пересечением двух совместимых по типу отношений *A* и *B* (*A intersect B*) называется отношение с тем же заголовком и телом, состоящим из множества кортежей *t*, принадлежащих одновременно обоим отношениям *A* и *B*.

$$C = (A\ intersect\ B) \mid \forall\ t_i \in C \mid t_i \in A\ \&\ t_i \in B.$$

Пример: *A* intersect *B* для нашего примера представляет детали, изготовленные из стали и весом более 0,5 кг.

K	Название детали	Вес	Материал
K1	*D1*	0.8	Сталь

K2	*D2*	1.0	Сталь

Вычитанием двух совместимых по типу отношений *A* и *B* (*A minus B*) называется отношение с тем же заголовком и телом, состоящим из множества кортежей *t*, принадлежащих отношению *A* и не принадлежащих отношению *B*.

$$C = (A\ minus\ B) \mid \forall\ t_i \in C \mid t_i \in A\ \&\ t_i \notin B.$$

Пример: выражение (*A* minus *B*) представляет детали, которые готовлены из стали и не весят больше 0,5 кг.

C = (A minus B)

K	Название детали	Вес	Материал
K3	*D3*	0.8	Сталь

C = (B minus A)

K	Название детали	Вес	Материал
K4	*D4*	0.7	Алюминий

Контрольные вопросы

1. История реляционной модели. Основная идея РМ.
2. Основная идея реляционной модели. Терминология реляционной модели.
3. Ключи реляционной модели. Простой и составной ключ
4. Какая модель данных считается реляционной.
5. Опишите основные элементы реляционной таблицы в примере.
6. Как устанавливается взаимосвязь между таблицами в реляционной модели данных? Приведите пример.
7. Дайте описание к следующим терминам реляционной модели: кортеж, домен, атрибут.
8. Что такое первичный и внешний ключ?
9. Основные операторы реляционной алгебры?

6. НОРМАЛИЗАЦИЯ БАЗЫ. ФУНКЦИОНАЛЬНАЯ ЗАВИСИМОСТЬ, АТРИБУТА. КЛЮЧ. ПЕРВАЯ, ВТОРАЯ, ТРЕТЬЯ И ДРУГИЕ НОРМАЛЬНЫЕ ФОРМЫ СХЕМ ОТНОШЕНИЙ. АЛГОРИТМ ПЕРЕХОДА К ТРЕТЬЕМУ НОРМАЛЬНОЙ ФОРМЕ. ПРИМЕРЫ

6.1. Процесс нормализации базы данных

Основной задачей **логического проектирование** является разработка логической схемы, ориентированной на СУБД. На этом этапе определяются отношения реляционной модели (таблицы СУБД), атрибуты (столбцы таблиц) и типы атрибутов (типы данных столбцов).

Выделяется несколько основных методов логического проектирования:

- отображение ER-диаграммы на логическую схему;
- нормализация таблиц.

Нормализация - это процесс уменьшения избыточности информации базы данных.

Нормализация - это разбиение таблицы на две или больше, обладающих лучшими свойствами при добавлении, изменении и удалении данных.

Основная цель нормализации - получение такого проекта базы данных, в котором каждый факт появляется лишь в одном месте, т.е. исключена избыточность информации.

Это делается для исключения противоречивости хранимых данных

Нормализация таблиц является наиболее формализованным методом логического проектирования и при ее применении не требуется построения ER-диаграммы.

Процесс нормализации представляет собой переход от одной нормальной формы к следующей, причем каждая следующая нормальная форма обладает свойствами лучшими, чем предыдущая.

Каждой нормальной форме соответствует некоторый определенный набор ограничений, и отношение находится в некоторой нормальной форме, если удовлетворяет свойственному ей набору ограничений.

В теории реляционных баз данных выделяется следующая последовательность нормальных форм:

- первая нормальная форма (1NF);
- вторая нормальная форма (2NF);
- третья нормальная форма (3NF);
- нормальная форма Бойса-Кодда (BCNF);
- четвертая нормальная форма (4NF);
- пятая нормальная форма, или нормальная форма проекции-соединения (или PJ/NF).

Основные свойства нормальных форм:

- каждая следующая нормальная форма в некотором смысле лучше предыдущей;

- при переходе к следующей нормальной форме свойства предыдущих нормальных форм сохраняются.

6.2. Проблемы при построении БД

Прежде чем мы приступим к обсуждению методов проектирования хороших схем баз данных, давайте посмотрим, почему некоторые схемы могут оказаться неадекватными. В частности, обратимся к схеме отношения

Поставщики(назв_пост, адрес_пост, товар, цена)

НАЗВ_ПОСТ	АДРЕС_ПОСТ	ТОВАР	ЦЕНА
АГАТА-ИМПЕКС	ул Ата-тюрк 208	Компьютер P IV	2 000 000
НУРОН	ул Навои 158	Монитор LCD 17»	254 000
TS-TECHNOLOGY	ул Янгиюльская 64	Компьютер P IV 1	800 000
НУРОН	ул Навои 158	Клавиатура	25 000
SHARIFA-T	Чиланзар-б 64	Мышь	15 000

В связи с этой схемой возникает несколько проблем:

1. **Избыточность**. Адрес поставщика повторяется для каждого поставляемого товара.

2. **Потенциальная противоречивость(аномалии обновления)**. Вследствие избыточности мы можем обновлять адрес поставщика в одном кортеже, оставляя его неизменным в другом. Таким образом, может оказаться, что для некоторых поставщиков нет единого адреса. Однако интуитивно мы чувствуем, что он должен быть.

3. **Аномалии включения**. В базу данных не может быть записан адрес поставщика, если он в настоящее время не поставляет по меньшей мере один товар. Можно, конечно, поместить неопределенные значения в компоненты ТОВАР и ЦЕНА кортежа для этого поставщика. Но если он начнет поставлять некоторый товар, не забудем ли мы удалить кортеж с неопределенными значениями? Хуже того, ТОВАР и НАЗВ_ПОСТ образуют ключ данного отношения, и поиск кортежей с неопределенными значениями в ключе может быть затруднительным или невозможным.

4. **Аномалии удаления**. Обратная проблема возникает при необходимости удаления всех товаров, поставляемых данным поставщиком, вследствие чего мы непреднамеренно утрачиваем его адрес.

Возникает вопрос: « Как найти хорошую замену для плохой схемы отношений?»

2 Нормальные формы.

Отношения можно классифицировать по типам аномалий, модификации, которым они подвержены. В 1970-х годах теоретики реляционных баз данных, постепенно собирали количество этих типов. Кто-то назвал аномалию, классифицировал ее и думал, как предотвратить ее возникновение. Каждый раз, когда это происходило, критерии построения отношений совершенствовались. Эти классы отношений и способы предотвращения аномалий называются нормальными формами(normal forms). В зависимости от своей структуры, отношение может быть в первой, во второй или в какой-либо другой нормальной форме.

В своей работе, последовавшей за знаковой статьей 1970 г., Кодд и другие определили первую, вторую и третью нормальные формы(1НФ, 2НФ и 3НФ). Позднее была введена нормальная форма Бойса-Кодда(НФБК), а затем были определены четвертая и пятая нормальные формы. Как показывает рис. 5.6, эти нормальные формы являются вложенными. То есть отношение во второй нормальной форме является также отношением в первой нормальной форме, а отношение в5НФ(пятая нормальная форма) находится одновременно в4НФ, НФБК, 3НФ, 2НФ и1НФ.

Эти нормальные формы помогали, но у них было и серьезное ограничение. Не было теории, гарантирующей, что какая-либо из этих форм устранит все аномалии: каждая форма могла устранить только определенные их виды. Эта ситуация разрешилась в1981 г., когда Р. Фагин(R. Fagin) ввел новую нормальную форму, которую он назвал доменно-ключевой нормальной формой, или ДКНФ(domain/key normal form, DK/NF). В своей важной статье Фагин показал, что отношение в ДКНФ свободно от всех аномалий модификации, независимо от их типа1. Он также показал, что любое отношение, свободное от аномалий модификации, должно находиться в ДКНФ.

До введения ДКНФ теоретикам реляционных баз данных приходилось продолжать поиск все новых и новых аномалий и нормальных форм. Доказательство Фагина упростило ситуацию. Если мы можем привести отношение к ДКНФ, можем быть уверены, что в нем не будет аномалий модификации. Вся загвоздка в том, как привести отношение к ДКНФ.

Основные свойства нормальных форм:

• каждая следующая нормальная форма в некотором смысле лучше предыдущей;

• при переходе к следующей нормальной форме свойства предыдущих нормальных свойств сохраняются.

В основе процесса проектирования лежит метод нормализации, декомпозиция отношения, находящегося в предыдущей нормальной форме, в одно или более отношения, удовлетворяющих требованиям следующей нормальной формы.

6.3 Нормализация и функциональные зависимости

Нормализация— это процесс преобразования отношения, имеющего некоторые недостатки, в отношение, которое этих недостатков не имеет. Что еще более важно, нормализацию можно использовать как критерий для определения желательности и правильности отношений. Вопрос о том, что такое хорошо структурированное отношение, был предметом многочисленных теоретических исследований. Термин нормализация обязан своим появлением одному из пионеров технологии баз данных, Э. Ф. Кодду(E. F. Codd), который определил различные нормальные формы(normal forms) отношений.

Наиболее важные на практике нормальные формы отношений основываются на фундаментальном в теории реляционных баз данных понятии функциональной зависимости. Для дальнейшего изложения нам потребуются несколько определений.

Функциональная зависимость(functional dependency)— это связь между атрибутами. Предположим, что если мы знаем значение одного атрибута, то можем вычислить(или найти) значение другого атрибута. Например, если нам известен номер счета клиента, то мы можем определить состояние его счета. В таком случае мы можем сказать, что атрибут СостояниеСчетаКлиента функционально зависит от атрибута НомерСчетаКлиента.

Говоря более общим языком, атрибутY функционально зависит от атрибутаX, если значениеX определяет значениеY. Другими словами, если нам известно значениеX, мы можем определить значениеY.

Уравнения выражают функциональные зависимости. Например, если мы знаем цену и количество приобретенного товара, мы можем определить стоимость покупки по следующей формуле:

Стоимость= Цена x Количество

В этом случае мы могли бы сказать, что атрибут Стоимость функционально зависит от атрибутов Цена и Количество.

Функциональные зависимости между атрибутами в отношении обычно не выражаются уравнениями. Пусть, например, каждому студенту присвоен уникальный идентификационный номер, и у каждого студента есть одна и только одна специальность.

Имея номер студента, мы можем узнать его специальность, поэтому атрибут Специальность функционально зависит от атрибута НомерСтудента. Или рассмотрим компьютеры в вычислительной лаборатории. Каждый компьютер имеет конкретный размер основной памяти, поэтому атрибут ОбъемПамяти функционально зависит от атрибута СерийныйНомерКомпьютера.

В отличие от случая с уравнением, такие функциональные зависимости нельзя разрешить при помощи арифметики; вместо этого они хранятся в базе

Ключ(key) — это группа из одного или более атрибутов, которая уникальным образом идентифицирует строку. Каждое отношение имеет минимум один ключ. Это утверждение должно быть верным, поскольку ни одно отношение не может иметь одинаковых строк, и, следовательно, в крайнем случае, ключ будет состоять из всех атрибутов отношения.

6.4 Первая нормальная форма

О любой таблице данных, удовлетворяющей определению отношения, говорят, что она находится в первой нормальной форме(first normal form, INF). Вспомните, что для того, чтобы таблица была отношением, должно выполняться следующее: ячейки таблицы должны содержать одиночные значения и в качестве значений не допускаются ни повторяющиеся группы, ни массивы. Все записи в одном столбце(атрибуте) должны иметь один и тот же тип. Каждый столбец должен иметь уникальное имя, но порядок следования столбцов в таблице несуществен. Наконец, в таблице не может быть двух одинаковых строк, и порядок следования строк несуществен.

Первая нормальная форма:

Таблица находится в **первой нормальной форме (1НФ)** тогда и только тогда, когда ни одна из ее строк не содержит в любом своем поле более одного значения и ни одно из ее ключевых полей не пусто (NULL значение).

Первая нормальная форма (1НФ). Отношение, в котором на пересечении каждой строки и каждого столбца содержится одно и только одно значение.

Перед обсуждением первой нормальной формы целесообразно дать определение того состояния, которое ей предшествует.

Ненормализованная форма (ННФ). Таблица, содержащая одну или несколько повторяющихся групп данных.

Понятие повторяющейся группы. Повторяющаяся группа (repeating group) это атрибут, имеющий несколько значений в каждой строке.

Пример. Предположим, что в отношении служащих нужно хранить имена и даты рождения их детей. У каждого служащего может быть по несколько детей, поэтому имена детей и даты их рождения образуют повторяющуюся группу.

Emp.ID	First	Last	Children's Names	Children's Birthdates
1001	Jane	Doe	Macy, Sam	1/1/92,5/15/94
1002	John	Doe	Mary, Sam	2/2/92,5/10/94
1003	Jane	Smith	John, Pat, Lee, Macy	10/5/94,10/12/90,6/6/96,8/21/94
1004	John	Smith	Michael	7/4/96
1005	Jane	Jones	Edward, Martha	10/21/95, 10/15/89

Рис.1. Отношение с повторяющимися группами

Обратите внимание: в одной строке здесь присутствует по несколько значений в столбцах имен детей (Children's Names) и дат их рождения (Children's Birthdates). Возникают две основные проблемы:

- Нет Возможности точно сказать, какие даты рождения каким детям соответствуют. Предположим, что мы можем сопоставить даты рождения именам детей по их позиции в списке, однако нет уверенности в том, что номера позиций всегда будут соблюдаться.
- Поиск в такой таблице чрезвычайно труден. Например, если необходимо узнать, дети каких сотрудников родились до 1995 с., СУБД потребуется извлечь даты рождения из столбца дат рождения, а лишь затем проверить их. Если же невозможно узнать, сколько дат рождения находится в этом столбце для конкретной строки, объем обработки информации становится еще больше.

Устранение повторяющихся групп

Существуют два способа устранения повторяющихся групп и приведения отношения в соответствие требованиям первой нормальной формы: верный и неверный.

1. При первом (неверный) способе повторяющиеся группы устраняются путем ввода соответствующих данных в пустые столбцы строк с

повторяющимися данными. Иначе говоря, пустые места при этом заполняются дубликатами неповторяющихся данных. Этот способ часто "выравниванием" ("flattening") таблицы. Полученная в результате этих действий таблица, которая теперь будет называться отношением, содержит элементарные (или единственные) значения на пересечении каждой строки с каждым столбцом и поэтому находится в первой нормальной форме. В результате применения такого способа в полученное отношение вносится определенная избыточность данных, которая в ходе дальнейшей нормализации будет устранена.

Пример. Сначала рассмотрим неверный способ, чтобы знать, чего именно не следует делать. На рис. 2 приведено отношение, где повторяющиеся группы устраняются созданием нескольких столбцов для нескольких значении. В этом конкретном примере три пары столбцов имен и дат рождения детей.

Emp.ID	First	Last	Child Name1	Child Bdate1	Child Name2	Child Bdate2	Child Name3	Child Bdate3
1001	Jane	Doe	Macy	Sam		1/1/92	5/15/94	
1002	John	Doe	Mary	Sam		2/2/92,	5/10/94	
1003	Jane	Smith	John	Pat	Lee	10/5/94	10/12/90	6/6/96
1004	John	Smith	Michael			7/4/96		
1005	Jane	Jones	Edward	Martha		10/21/95	10/15/89	

Рис. 2. Неверное устранение повторяющихся групп

Отношение, показанное на рис. 2, вполне отвечает критериям первой нормальной формы: повторяющихся групп больше нет, и проблемы определения соответствия дат рождения не существует. Однако в такой структуре возникает ряд собственных проблем:

• Отношение ограничено тремя детьми для каждого служащего. Это означает, что для хранения сведении о четвертом ребенке Джейн Смит (Jane Smith) места нет. Может быть, нужно создать для Джейн Смит дополнительную строку в таблице? Если это сделать, то первичный ключ отношения не сможет и дальше быть идентификатором служащего (Emp.ID). В состав первичного ключа придется внести еще как минимум имя одного ребенка.

• Пространство этого отношения понапрасну расходуется на тех людей, у которых не больше трех детей. При условии, что дисковое пространство является одним из самых недорогих элементов системы базы данных, это, пожалуй, наименее важная из всех проблем, ..связанных с рассматриваемым отношением.

• Поиск сведений о конкретном ребенке становится очень громоздким. Чтобы ответить на вопрос: "Есть ли у кого-то ребенок по имени Ли?", СУБД приходится формировать запрос, в котором просматриваются все три столбца имен детей, так как узнать, в каком именно столбце можно найти нужное имя, нельзя.

2. При втором (верный) способе один атрибут или группа атрибутов назначаются ключом ненормализованной таблицы, а затем повторяющиеся группы изымаются и помещаются в отдельные отношения вместе с копиями ключа исходной таблицы. Далее в новых отношениях устанавливаются свои первичные ключи. Иногда ненормализованная таблица может содержать несколько повторяющихся групп или включать повторяющиеся группы, содержащиеся в других повторяющихся группах. В таких случаях данный прием применяется до тех пор, пока повторяющихся групп совсем не останется. Полученный набор отношений будет находиться в первой нормальной форме только тогда, когда ни в одном из них не будет повторяющихся групп атрибутов.

Пример. Правильный способ устранения повторяющихся групп создание еще одной таблицы (еще одной сущности) для работы с несколькими экземплярами повторяющей я группы. В рассматриваемом примере можно создать для детей (Children) вторую таблицу, которая выглядит примерно так, как показано на рис. 3.

Employees

Emp.ID	First	Last
1001	Jane	Doe
1002	John	Doe
1003	Jane	Smith
1004	John	Smith
1005	Jane	Jones

Children

Emp.ID	Child Name	Birthdate
1001	Macy	1/1/92
1001	Sam	5/15/94
1002	Mary	2/2/92
1002	Sam	5/10/94
1003	John	10/5/94

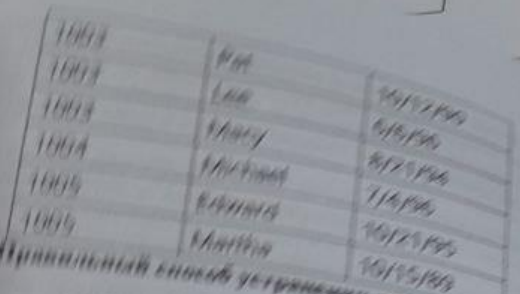


Рис. 4. Правильный способ устранения повторяющейся группы

6.5 Вторая нормальная форма

Отношение находится во второй нормальной форме, если все его неключевые атрибуты зависят от всего ключа. В соответствии с этим определением, если отношение имеет в качестве ключа одиночный атрибут, то оно автоматически находится во второй нормальной форме.

Поскольку ключ является одиночным атрибутом, то по умолчанию любой неключевой атрибут зависит от всего ключа, и частичных зависимостей быть не может. Таким образом, вторая нормальная форма представляет интерес только для тех отношений, которые имеют композитные ключи.

6 Третья нормальная форма.

Отношения во второй нормальной форме также могут иметь аномалии. Смотрим отношение ПРОЖИВАНИЕ на рис. 2. Ключом здесь является НомерСтудента, и имеются функциональные зависимости НомерСтудента> Общежитие и Общежитие> Плата. Эти зависимости возникают потому, что каждый студент живет только в одном общежитии, и каждое общежитие берет со всех проживающих в нем студентов одинаковую плату. Например, каждый живущий в общежитии Рэндольф-Холл платит$3200 за квартал.

Функциональные зависимости:Общежитие → Плата
НомерСтудента → Общежитие → Плата

НомерСтудента	Общежитие	Плата
100	Рэндальф	3200
150	Ингерсол	3100
200	Рэндальф	3200
250	Питкин	3100
300	Рэндальф	3200

Рисунок 2

Поскольку НомерСтудента определяет атрибут Общежитие, а Общежитие определяет атрибут Плата, то косвенным образом НомерСтудента> Плата. Такая структура функциональных зависимостей называется транзитивной зависимостью (transitive dependence), поскольку атрибут НомерСтудента определяет атрибут Плата через атрибут Общежитие.

Ключом отношения ПРОЖИВАНИЕ является НомерСтудента, который является одиночным атрибутом, и, следовательно, отношение находится во второй нормальной форме(и Общежитие, и Плата определяются атрибутом НомерСтудента). Несмотря на это, отношение ПРОЖИВАНИЕ имеет аномалии, обусловленные транзитивной зависимостью.

Мы потеряем не только тот факт, что студент №150 живет в Ингерсолл-Холле, но и тот факт, что проживание в этом общежитии стоит$3100. Это аномалия удаления. А как мы можем записать тот факт, что плата за проживание в Кэрригт-Холле составляет$3500? Никак, пока туда не решит вселиться хотя бы один студент. Это аномалия вставки.

Что произойдет, если мы удалим вторую строку отношения на рис. 2?

НомерСтудента	Общежитие
100	Рэндольф
150	Ингерсол
200	Рэндольф
250	Питкин
300	Рэндольф

Общежитие	Плата
Рэндольф	3200
Ингерсол	3100
Рэндольф	3200
Питкин	3100
Рэндольф	3200

Рисунок 3

Чтобы удалить аномалии из отношения во второй нормальной форме, необходимо устранить транзитивную зависимость рис. 3. Отношение находится в третьей нормальной форме, если оно находится во второй нормальной форме и не имеет транзитивных зависимостей.

6.6. Нормальная форма Бойса-Кодда

Поскольку студенты могут специализироваться в нескольких областях, атрибут НомерСтудента не определяет атрибут Специальность. Более того, так как студент может иметь несколько консультантов, НомерСтудента не определяет и атрибут Преподаватель. Таким образом, НомерСтудента сам по себе не может быть ключом.

Комбинация(НомерСтудента, Специальность) определяет атрибут Преподаватель, а комбинация(НомерСтудента, Преподаватель) определяет атрибут Специальность. Следовательно, любая из этих комбинаций может быть ключом. Два или более атрибута или группы атрибутов, которые могут

быть ключом, называются ключами-кандидатами (candidate keys). Тот из ключей-кандидатов, который выбирается в качестве ключа, называется первичным ключом(primary key).

Функциональные зависимости: НомерСтудента Специальность → Преподаватель

НомерСтудента	Специальность	Преподаватель
100	Математика	Коши
150	Психология	Юнг
200	Математика	Риман
250	Математика	Коши
300	Рэндольф	Перлс
300	Психология	Риман

Рисунок 4

Кроме ключей-кандидатов, есть еще одна функциональная зависимость, которую следует рассмотреть: атрибут Преподаватель определяет атрибут Специальность(любой из преподавателей является консультантом только по одному предмету; следовательно, зная имя преподавателя, мы можем определить специальность). Таким образом, Преподаватель является детерминантом.

По определению, отношение КОНСУЛЬТАНТ находится в первой нормальной форме. Оно также находится во второй нормальной форме, поскольку не имеет неключевых атрибутов(каждый из атрибутов является частью минимум одного ключа). Наконец, это отношение находится в третьей нормальной форме, так как не имеет транзитивных зависимостей. Тем не менее, несмотря на все это, отношение имеет аномалии модификации.

Пусть студент с номером300 отчисляется из университета. Если мы удалим строку с информацией о студенте с номером300, мы потеряем тот факт, что Перле является консультантом по психологии. Это аномалия удаления. Далее, как мы можем записать в базу тот факт, что Кейнс является консультантом по экономике? Никак, пока не появится хотя бы один студент, специализирующийся на экономике. Это аномалия удаления.

Ситуации, подобные только что описанной, приводят нас к определению нормальной формы Бойса-Кодда(Boyce-Codd normal form, BK/NF): отношение

находится в НФБК, если каждый детерминант является ключом-кандидатом. Отношение КОНСУЛЬТАНТ не находится в НФБК, поскольку детерминант Преподаватель не является ключом-кандидатом.

НомерСтудента	Преподаватель
100	Коши
150	Юнг
200	Риман
250	Коши
300	Перлс
300	Риман

Консультант	Предмет
Коши	Математика
Юнг	Психология
Риман	Математика
Перлс	Психология

Рисунок 5

Как и в других примерах, отношение КОНСУЛЬТАНТ можно разбить на два отношения, не имеющие аномалий. Например, отношения СТУДЕНТ-КОНСУЛЬТАНТ (НомерСтудента, Преподаватель) и КОНСУЛЬТАНТ-ПРЕДМЕТ(Преподаватель, Специальность) не имеют аномалий рис. 5.

6.7. Четвертая нормальная форма

Многозначные зависимости: НомерСтудента → → Специальность
НомерСтудента → → Секция

НомерСтудента	Специальность	Секция
100	Музыка	Плавание
100	Бухгалтерский учет	Плавание
100	Музыка	Теннис
100	Бухгалтерский учет	Теннис
150	Математика	Оздоровительный бег

Рис. 5.9. Отношение с многозначными зависимостями

Рисунок 6

Рассмотрим отношение СТУДЕНТ на рис. 6, которое отображ между студентами, специальностями и секциями. Предположим, что могут иметь несколько специальностей и заниматься в нескольких секциях. В таком случае единственным ключом

комбинация(НомерСтудента, Специальность, Секция). Например, студентка с номером100 специализируется на музыке и бухгалтерском учете и, кроме того, посещает секции плавания и тенниса, а студент с номером150 специализируется только на математике и занимается бегом.

Какова связь между атрибутами НомерСтудента и Специальность? Это не функциональная зависимость, поскольку у студента может быть несколько специальностей. Одному и тому же значению атрибута НомерСтудента может соответствовать много значений атрибута Специальность. Помимо того, одному и тому же значению атрибута НомерСтудента может соответствовать много значений атрибута Секция.

Такая зависимость атрибутов называется многозначной зависимостью(multivalue dependency). Многозначные зависимости приводят к аномалиям модификации. Для начала обратите внимание на избыточность данных на рис. 6. Студентке с номером100 посвящено четыре записи, в каждой из которых указана одна из ее специализаций и одна из посещаемых ею секций. Если бы те же данные хранились в меньшем количестве строк (скажем, было бы две строки— одна для музыки и плавания, а другая для бухгалтерского учета и тенниса), это дезориентировало бы пользователей. Получалось бы, что студентка с номером100 плавает только тогда, когда специализируется на музыке, а в теннис играет только тогда, когда специализируется на бухгалтерском учете. Но такая интерпретация нелогична. Специальности и секции совеершенно независимы друг от друга. Поэтому, чтобы избежать таких неверных заключений, мы храним все сочетания специальностей и секций.

НомерСтудента	Специальность	Секция
100	Музыка	Лыжи
100	Музыка	Плавание
100	Бухгалтерский учет	Плавание
100	Музыка	Теннис
100	Бухгалтерский учет	Теннис
150	Математика	Оздоровительный бег

а

НомерСтудента	Специальность	Секция
100	Музыка	Лыжи
100	Бухгалтерский учет	Лыжи
100	Музыка	Плавание
100	Бухгалтерский учет	Плавание
100	Музыка	Теннис
100	Бухгалтерский учет	Теннис
150	Математика	Оздоровительный бег

б

Рисунок 7

Допустим, что студентка с номером100 решила записаться в секцию к, и поэтому мы добавляем в таблицу строку[100, Музыка, Лыжи], как зано на рис. 7 а данный момент из отношения можно сделать вывод, что ентка100 занимается лыжами только как музыкант, но не как бухгалтер. ы данные имели согласованный характер, мы должны добавить столько сколько имеется специальностей, и в каждой из них указать секцию Таким образом, мы должны добавить строку[100, Бухгалтерский учет,

Лыжи], как показано на рис. 7 б. Это аномалия обновления: требуется слишком много модификаций, чтобы внести одно простое изменение.

Вообще говоря, многозначная зависимость существует, когда отношение имеет минимум три атрибута, причем два из них являются многозначными, а их значения зависят только от третьего атрибута. Другими словами, в отношенииR (A, B, C) существует многозначная зависимость, если А многозначным образом определяет В и С, а сами В и С не зависят друг от друга. Как мы видели из предыдущего примера, НомерСтудента многозначно определяет атрибуты Специальность и Секция, но сами Специальность и Секция не зависят друг от друга.

Чтобы устранить эти аномалии, мы должны избавиться от многозначной зависимости. Мы сделаем это, создав два отношения, в каждом из которых будут храниться данные только по одному многозначному атрибуту. Результирующие отношения не будут иметь аномалий. Это отношения СТУДЕНТ-СПЕЦИАЛЬНОСТЬ (НомерСтудента, Специальность) и СТУДЕНТ-СЕКЦИЯ(НомерСтудента, Секция), приведенные на рис8.

НомерСтудента	Специальность
100	Музыка
100	Бухгалтерский учет
150	Математика

НомерСтудента	Секция
100	Лыжи
100	Плавание
100	Теннис
150	Оздоровительный бег

Рис. 5.11. Устранение многозначной зависимости

Рисунок 8

Отношение находится в четвертой нормальной форме, если оно находится в НФБК и не имеет многозначных зависимостей.

6.8. Пятая нормальная форма

Пятая нормальная форма(fifth normal form, 5NF) связана с зависимостями, которые имеют несколько неопределенный характер. Речь здесь идет об отношениях, которые можно разделить на несколько более мелких отношений, как мы это делали выше, но затем невозможно восстановить.

Условия, при которых возникает эта ситуация, не имеют ясной, интуитивной интерпретации. Нам неизвестно, каковы следствия таких зависимостей; мы не знаем даже, есть ли у них какие-либо практические следствия.

10 Доменно-ключевая нормальная форма.

В1981 г. Фагин опубликовал важную статью, в которой он определи доменно-ключевую нормальную форму(domain/key normal form, DKNFI.

показал, что отношения в ДКНФ не имеют аномалий модификации и, более того, любое отношение, не имеющее аномалий модификации, находится в ДКНФ.

Это открытие положило конец введению нормальных форм, и теперь в нормальных формах более высокого порядка нет необходимости — по крайней мере, для устранения аномалий модификации.

Контрольные вопросы

1. Основная задача логического проектирования БД.
2. Что такое нормализация баз данных?
3. В чём состоит основная цель нормализации БД?
4. Какая таблица называется нормализованной?
5. Основные виды зависимостей между полями реляционной таблицы.
6. Нормальные формы (1НФ, 2НФ, 3НФ, НФБК, 5НФ) реляционных отношений.
7. Как осуществляется построение рационального варианта схем отношений?
8. Первая нормальная форма. Какая таблица находится в первой нормальной форме?
9. Понятие повторяющейся группы в ненормализованной таблице. Приведите пример к ненормализованной таблице.
10. Способы устранение повторяющихся групп в ненормализованной таблице.
11. Неверный способ устранение повторяющихся групп в ненормализованной таблице. Приведите пример.
12. Верный способ устранение повторяющихся групп в ненормализованной таблице. Приведите пример.

7. SQL-СТРУКТУРИРОВАННЫЙ ЯЗЫК ЗАПРОСОВ. СТАНДАРТЫ SQL. СТРУКТУРА. ТИПЫ ДАННЫХ В SQL. ПРОСТЫЕ ЗАПРОСЫ SELECT

7.1. История развития SQL

SQL (Structured Query Language) — Структурированный Язык Запросов — стандартный язык запросов по работе с реляционными БД.

Как утверждалось в главе 3, история реляционной модели данных (и косвенно языка SQL) началась в 1970 году с публикации основополагающей статьи Е. Ф. Кодда (в то время он работал в исследовательской лаборатории корпорации IBM в Сан-Хосе). В 1974 году Д. Чемберлен, работавший в той же лаборатории, публикует определение языка, получившего название "Structured English Query Language", или SEQUEL. В 1976 году была выпущена переработанная версия этого языка, SEQUEL/2; впоследствии его название пришлось изменить на SQL по юридическим соображениям — аббревиатура SEQUEL уже использовалась кем-то ранее. Но до настоящего времени многие по-прежнему произносят аббревиатуру SQL как "сиквэл", хотя официально ее рекомендуется читать как "эс-кю-эл".

Язык SQL появился после реляционной алгебры, и его прототип был разработан в конце 70-х годов в компании IBM Research. Он был реализован в первом прототипе реляционной СУБД фирмы IBM System R. В дальнейшем этот язык применялся во многих коммерческих СУБД и в силу своего широкого распространения постепенно стал неофициальным стандартом для языков манипулирования данными в реляционных СУБД. Первый международный стандарт языка SQL был принят в 1989 г. Подавляющее большинство доступных на рынке СУБД поддерживают этот стандарт полностью. Однако развитие информационных технологий, связанных с базами данных, и необходимость реализации переносимых приложений потребовали в скором времени доработки и расширения первого стандарта SQL.

В конце 1992 г. был принят новый международный стандарт языка SQL (SQL/92 или SQL2). И он не лишен недостатков, но в то же время является существенно более точным и полным, чем SQL/89. В настоящий момент большинство производителей СУБД внесли изменения в свои продукты так, чтобы они в большей степени удовлетворяли стандарту SQL2.

В 1999 году появился новый стандарт, названный SQL3. Если отличия между стандартами SQL1 и SQL2 во многом были количественными, то стандарт SQL3 соответствует качественным серьезным преобразованиям. В SQL3 введены новые типы данных, при этом предполагается возможность задания сложных структурированных типов данных, которые в большей степени соответствуют объектной ориентации. SQL нельзя в полной мере отнести к традиционным языкам программирования, он не содержит

традиционные операторы, управляющие ходом выполнения программы, операторы описания типов и многое другое, он содержит только набор стандартных операторов доступа к данным, хранящимся в базе данных.

Язык SQL относительно прост в изучении.

- Это непроцедурный язык, поэтому в нем необходимо указывать, какая информация должна быть получена, а не как ее можно получить. Иначе говоря, язык SQL не требует указания методов доступа к данным, • Как и большинство современных языков, SQL поддерживает свободный формат записи операторов. Это означает, что при вводе отдельные элементы операторов не связаны с фиксированными позициями на экране.
- Структура команд задается набором ключевых слов, представляющих собой обычные слова английского языка, такие как CREATE TABLE (Создать таблицу), INSERT (Вставить), SELECT (Выбрать). Например:

CREATE TABLE Staff (staffNo VARCHAR(S), IName VARCHAR(15), salary DECIMAL(7,2));

7.2. Назначения и стандарты SQL

SQL (Structured Query Language) – непроцедурный язык взаимодействия приложений и пользователей с реляционными СУБД, состоящий из набора стандартных команд на английском языке.

Отдельные команды изначально никак логически не связаны друг с другом.

***Язык SQL (Structured Query Language – структурированный язык запросов) применяется для общения пользователя с реляционной базой данных* и состоит из следующих частей**:

- операторы определения объектов базы данных (Data Definition Language – **DDL**): *CREATE, ALTER, DROP* и тд;
- операторы манипулирования данными (Data Manipulation Language - **DML**): INSERT, UPDATE, DELETE, SELECT;
- операторы защиты и управления данными (Data Control Language – **DCL**): GRANT, REVOKE.
- команды управления транзакциями (Transaction Control Language – **TCL**): COMMIT, ROLLBACK, SAVEPOINT;

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, такие как ставка, модификация и удаление данных из таблиц;
- выполнять простые и сложные запросы.

Кроме того, язык работы с базами данных должен решать все указанные ше задачи при минимальных усилиях со стороны пользователя, а структура интаксис его команд должны быть 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.

Год	Название	Иное название	Изменения
1986	SQL-86	SQL-87	Первый вариант стандарта, принятый институтом ANSI и одобренный ISO в 1987 году.
1989	SQL-89	FIPS 127-1	Немного доработанный вариант предыдущего стандарта.
1992	SQL-92	SQL2, FIPS 127-2	Значительные изменения (ISO 9075); уровень Entry Level стандарта SQL-92 был принят как стандарт FIPS 127-2.
1999	SQL:1999	SQL3	Добавлена поддержка регулярных выражений, рекурсивных запросов, поддержка триггеров, базовые процедурные расширения, нескалярные типы данных и некоторые объектно-ориентированные возможности.
2003	SQL:2003		Введены расширения для работы с XML-данными, оконные функции (применяемые для работы с OLAP-базами данных), генераторы последовательностей и основанные на них типы данных.
2006	SQL:2006		Функциональность работы с XML-данными значительно расширена. Появилась возможность совместно использовать в запросах SQL и XQuery.
2008	SQL:2008		Улучшены возможности оконных функций, устранены некоторые неоднозначности стандарта SQL:2003[7]
2011	SQL:2011		Реализована поддержка хронологических баз данных (PERIOD FOR), поддержка конструкции FETCH[8].
2016	SQL:2016		Защита на уровне строк, полиморфные табличные функции, JSON.

7.3. Разновидности и режими SQL

До появления стандарта SQL3 язык SQL включал только команды определения и манипулирования данными; в нем отсутствовали какие-либо команды управления ходом вычислений. Другими словами, в этом языке не было команд IF ... THEN ...ELSE, GO TO, DO ... WHILE и любых других, предназначенных для управления ходом вычислительного процесса.

Подобные задачи должны были решаться программным путем (с помощью языков программирования или управления заданиями) либо интерактивно (в результате действий, выполняемых самим пользователем). По причине подобной незавершенности (с точки зрения организации вычислительного процесса) язык SQL мог использоваться двумя способами. Первый предусматривал *интерактивную* работу, заключающуюся во вводе пользователем с терминала отдельных операторов SQL.

Существуют и используются две формы языка SQL: интерактивный SQL и режим выполнения прикладных программ (приложений).

В интерактивном режиме работы с базой данных: пользователь работает с базой данных в прямом диалоге: вводит запрос на языке SQL – получает результат, вводит другой запрос – получает другой результат и т.д.

Второй состоял во *внедрении* операторов SQL в программы на процедурных языках. Язык SQL3, формальное определение которого принято в 1999 году.

Встроенный SQL представляется операторами языка SQL, встроенные в прикладные программы, написанные на других языках программирования (в других программных средах). Это дает возможность работы с базой данных с помощью прикладных программ, написанных на других алгоритмических языках, но требует включения дополнительных средств, обеспечивающих интерфейс между операторами языка SQL и соответствующим языком программирования.

7.4. Терминология

Под запросом, реализуемым с помощью языка SQL- запросов к базе данных, понимается команда, предназначенная для выполнения (и выполняемая) системой управления базами данных определяемого этой командой действия с базой данных.

Запрос реализуется с помощью операторов языка SQL. Операторы состоят из отдельных логических частей, называемых предложениями. Стандарты языка SQL регламентируют синтаксис операторов.

Несмотря на то, что язык SQL работает с реляционной базой данных, вместо термина «отношение» здесь используется термин «таблица», вместо терминов «кортеж» и «атрибут» используются соответственно термины «строка» и «столбец».

5. Структура (операторы) SQL

SQL содержит разделы, представленные в таблице 11.1.

Таблица 1. Операторы определения данных DDL(Data Definition Language — язык описания данных)

Оператор	Смысл	Действие
CREATE TABLE	Создать таблицу	Создает новую таблицу в БД
DROP TABLE	Удалить таблицу	Удаляет таблицу из БД
ALTER TABLE	Изменить таблицу	Изменяет структуру существующей таблицы или ограничения целостности, задаваемые для данной таблицы
CREATE VIEW	Создать представление	Создает виртуальную таблицу, соответствующую некоторому SQL-запросу

Таблица 2. Операторы манипулирования данными Data Manipulation Language (DMP)

Оператор	Смысл	Действие
DELETE	Удалить строки	Удаляет одну или несколько строк, соответствующих условиям фильтрации, из базовой таблицы. Применение оператора согласуется с принципами поддержки целостности, поэтому этот оператор не всегда может быть выполнен корректно, даже если синтаксически он записан правильно
INSERT	Вставить строку	Вставляет одну строку в базовую таблицу. Допустимы модификации оператора, при которых сразу несколько строк могут быть перенесены из одной таблицы или запроса в базовую таблицу
UPDATE	Обновить строку	Обновляет значения одного или нескольких столбцов в одной или нескольких строках, соответствующих условиям фильтрации

Таблица 3. Язык запросов Data Query Language (DQL)

Оператор	Смысл	Действие
SELECT	Выбрать строки	Оператор, заменяющий все операторы реляционной алгебры и позволяющий сформировать результирующее отношение, соответствующее запросу

Таблица 4. Средства управления транзакциями

Оператор	Смысл	Действие
COMMIT	Завершить транзакцию	Завершить комплексную взаимосвязанную обработку информации, объединенную в транзакцию
ROLLBACK	Откатить транзакцию	Отменить изменения, проведенные в ходе выполнения транзакции
'SAVEPOINT	Сохранить промежуточную точку выполнения транзакции	Сохранить промежуточное состояние БД, пометить его для того, чтобы можно было в дальнейшем к нему вернуться

Таблица 5. Средства администрирования данных

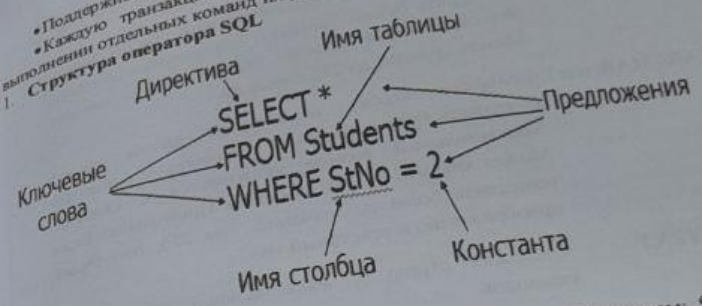
Оператор	Смысл	Действие
ALTER DATABASE	Изменить БД	Изменить набор основных объектов в базе данных, ограничений, касающихся всем базы данных
ALTER DBAREA	Изменить область хранения БД	Изменить ранее созданную область хранения
ALTER PASSWORD	Изменить пароль	Изменить пароль для всей базы данных
CREATE DATABASE	Создать БД	Создать новую базу данных, определив основные параметры для нее
CREATE DBAREA	Создать область хранения	Создать новую область хранения и сделать ее доступной для размещения данных
DROP DATABASE	Удалить БД	Удалить существующую базу данных (только в том случае, когда вы имеете право выполнить это действие)
DROP DBAREA	Удалить область хранения БД	Удалить существующую область хранения (если в ней на настоящий момент не располагаются активные данные)
GRANT	Предоставить права	Предоставить нрава доступа на ряд действий над некоторым объектом БД
REVOKE	Лишить прав	Лишить прав доступа к некоторому объекту или некоторым действиям над объектом

7.5. Синтаксис SQL

- Функции и названия объектов нечуствительны к регистру: SELECT = eCt.
- Однако при поиске по текстовым полям регистр учитывается
- SQL не чувствителен к переносу строк
- Отсутствуют обязательные символы, завершающие строки

- Поддерживаются --однострочные комментарии и /*многострочные */
- Каждую транзакцию принято завершать точкой с запятой, но при выполнении отдельных команд их употребление не обязательно.

1. **Структура оператора SQL**



Директивы описывают действие, выполняемое оператором: SELECT (выбрать), CREATE (создать), INSERT (добавить), DELETE (удалить), UPDATE (обновить), DROP (удалить), ALTER (изменить), COMMIT (завершить и зафиксировать внесенные изменения), ROLLBACK (отменить внесенные изменения).

Предложение описывает данные, с которыми работает оператор, или содержит уточняющую информацию о действии, выполняемом оператором: FROM (откуда), WHERE (где), GROUP BY (группировать по), HAVING (имеющий), ORDER BY (упорядочить по), INTO (куда).

2. Имена (идентификаторы)

– длина – до 128 символов

– используемые символы – только прописные или строчные буквы латинского алфавита, цифры или символ подчеркивания (_). *Первым символом* должна быть буква.

составное имя – идентификатор базы данных, ее владельца и (или) объекта базы данных. Например, *полное имя таблицы* состоит из имени владельца таблицы и имени таблицы, разделенных точкой (.): Admin.Students.

3. Комментарии

– /* и */ – многострочный комментарий

– -- – однострочный комментарий

4. Типы данных (MySQL).

Текстовые типы данных:

Тип данных	Описание
CHAR(size)	Содержит строку фиксированной длины (может содержать буквы, цифры и специальные символы). Фиксированный размер указывается в скобках. Может хранить до 255 символов
VARCHAR(size)	Содержит строку переменной длины (может содержать буквы, цифры и специальные символы). Максимальный размер указывается в скобках. Может хранить до 255 символов. Примечание: Если поместить большее значение, чем 255, оно будет преобразовано в текстовый тип
TINYTEXT	Содержит строку с максимальной длиной 255 символов
TEXT	Содержит строку с максимальной длиной 65 535 символов
BLOB	Для BLOB-объектов (двоичные большие объекты). Удерживает до 65 535 байт данных
MEDIUMTEXT	Содержит строку с максимальной длиной 16 777 215 символов
MEDIUMBLOB	Для BLOB-объектов (двоичные большие объекты). Удерживает до 16 777 215 байт данных
LONGTEXT	Содержит строку с максимальной длиной 4 294 967 295 символов
LONGBLOB	Для BLOB-объектов (двоичные большие объекты). Удерживает до 4 294 967 295 байт данных

Комплексные типы данных в MySQL:

Тип данных	Размер	Определение
ENUM(a,b,c,...,n)	1-255 значений: 1 байт 256-65,535 значений: 2 байта	Список. Максимальное количество значений в списке - 65,535. Поле может принимать только одно значение из списка. При неверном значении оставляет поле пустым. Пример поля: Gender ENUM("male","female").
SET(a,b,c,...,n)	1-8 значений: 1 байт 9-16 значений: 2 байта 17-24 значений: 3 байта 25-32 значений: 4 байта 33-64 значений: 8 байт	Список. Похож на ENUM, но поле может принимать несколько значений из списка. Максимальное количество значений в списке - 64. Пример поля: Fruits SET("orange","apple","kiwi").

Числовые типы данных:

Тип данных	Описание
TINYINT(size)	-128 до 127 нормальный. 0 до 255 неподписанный *. Максимальное количество цифр может быть указано в скобках
SMALLINT(size)	-32768 до 32767 нормальный. 0 до 65535 неподписанный *. Максимальное количество цифр может быть указано в скобках
MEDIUMINT(size)	-8388608 до 8388607 нормальный. 0 до 16777215 неподписанный *. Максимальное количество цифр может быть указано в скобках
INT(size)	-2147483648 до 2147483647 нормальный. 0 до 4294967295 неподписанный *. Максимальное количество цифр может быть указано в скобках
BIGINT(size)	-9223372036854775808 до 9223372036854775807 в норме. 0 для 18446744073709551615 неподписанных *. Максимальное количество цифр может быть указано в скобках
FLOAT(size,d)	Небольшое число с плавающей запятой. Максимальное количество цифр может быть указано в параметре size. Максимальное число цифр справа от десятичной запятой указано в параметре d

DOUBLE(size,d)	Большое число с плавающей запятой. Максимальное количество цифр может быть указано в параметре size. Максимальное число цифр справа от десятичной запятой указано в параметре d
DECIMAL(size,d)	Значение типа Double, хранящееся в виде строки, допускающее фиксированную десятичную точку. Максимальное количество цифр может быть указано в параметре size. Максимальное число цифр справа от десятичной запятой указано в параметре d

Типы данных дат:

Тип данных	**Описание**
DATE()	Свидание. Формат: гггг-мм-DD**Примечание:** Поддерживаемый диапазон от ' 1000-01-01 ' до ' 9999-12-31 '
DATETIME()	* комбинация даты и времени. Формат: гггг-мм-DD HH: MI: SS**Примечание:** Поддерживаемый диапазон от ' 1000-01-01 00:00:00 ' до ' 9999-12-31 23:59:59 '
TIMESTAMP()	* Временная метка. Значения timestamp хранятся в виде количества секунд со времени Unix (1970-01-01 00:00:00 ' UTC). Формат: гггг-мм-DD HH: MI: SS. **Примечание:** Поддерживаемый диапазон от ' 1970-01-01 00:00:01 ' UTC до ' 2038-01-09 03:14:07 ' UTC
TIME()	Время. Формат: HH: MI: SS **Примечание:** Поддерживаемый диапазон от '-838:59:59 ' до ' 838:59:59 '
YEAR()	Год в формате с двумя или четырьмя цифрами. **Примечание:**Допустимые значения в формате четырех цифр: 1901 до 2155. Допустимые значения в формате с двумя цифрами: 70 до 69, представляющие годы с 1970 по 2069

5. **Константы (литералы)**

- Числовые константы: 21, -345, +234,6547

Константы с плавающей запятой: 1.5E3, -3.14159E1, 2.5E7

- Строковые константы: 'Это символьная строка'.

Если в строковую константу нужно включить одинарную кавычку, то нее надо писать две одинарные кавычки:

Здесь внутри будут ' ' одинарные' ' кавычки'.

– Константы даты и времени. Пример для даты: '2012-10-03', '1993-12-10'. Пример для времени: '17:22:10', '01:01:01'.
– Логические константы: TRUE, FALSE, UNKNOWN
– Отсутствующие данные (значение NULL)

7.6. Операторы SQL

Операторы DDL - определения объектов базы данных :

- *CREATE DATABASE* - создать базу данных
- *DROP DATABASE* - удалить базы данных
- *CREATE TABLE* - создать таблицу
- *ALTER TABLE* - изменить таблицу
- *DROP TABLE* - удалить таблицу
- *CREATE DOMAIN* - создать домен
- *ALTER DOMAIN* - изменить домен
- *DROP DOMAIN* - удалить домен
- *CREATE VIEW* - создать представление
- *DROP VIEW* - удалить представление

Операторы DDL

•Операторы создания, изменения и удаления баз данных и объектов схемы данных

•Создание:

CREATE <OBJECT> <NAME> [параметры]

• Типы объектов:

– **DATABASE** – база данных;
– **SCHEMA** – схема данных;
– **TABLE** – таблица (отношение);
– **CONSTRAINT** – ограничение;
– **ATTRIBUTE** – атрибут;
– **VIEW** – представление;
– **INDEX** – индекс;
– **SEQUENCE** – последовательность;
– **STORED PROCEDURE** – хранимая процедура;
– **TRIGGER** – триггер;
– **USER** – пользователь БД.

Операторы DML - манипулирования данными

– *SELECT* - отобрать строки из таблиц
– *INSERT* - добавить строки в таблицу
– *UPDATE* - изменить строки в таблице

– *DELETE* - удалить строки в таблице

Команды управления транзакциями TCL

Используются для управления изменениями данных, производимыми DML-командами. С их помощью несколько DML-команд могут быть объединены в единое логическое целое, называемое транзакцией.

- *COMMIT* - завершить транзакцию и зафиксировать все изменения в БД.
- *ROLLBACK* - отменить транзакцию и отменить все изменения в БД.
- *SET TRANSACTION* - установить некоторые условия выполнения транзакции.

Операторы защиты и управления данными – DCL

- *GRANT* - предоставить привилегии пользователю или приложению на манипулирование объектами ;
- *REVOKE* - отменить привилегии пользователя или приложения.

Очень часто возникает необходимость произвести вычисление минимальных, максимальных или средних значений в столбцах. Так, например, может понадобиться вычислить средний балл. Для осуществления подобных вычислений SQL предоставляет специальные агрегатные функции:

MIN – минимальное значение в столбце;
MAX – максимальное значение в столбце;
SUM – сумма значений в столбце;
AVG – среднее значение в столбце;
COUNT – количество значений в столбце, отличных от NULL.

Следующий запрос считает среднее среди всех баллов, полученных студентами на экзаменах.

```
SELECT AVG(mark) FROM mark_st.
```

7.7. Встроенные функции

MySQL строковые функции

Функции	Описание
ASCII	Возвращает код числа, представляющий конкретный символ
R_LENGTH	Возвращает длину указанной строки (в символах)
ACTER_LEN GTH	Возвращает длину указанной строки (в символах)
ONCAT	Объединяет два или более выражений вместе

CONCAT_WS	Объединяет два или более выражений вместе и добавляет разделитель между ними
FIELD	Возвращает положение значения в списке значений
FIND_IN_SET	Возвращает положение строки в списке строк
FORMAT	Форматирует число как формат "#, # # #. # #", округляя его до определенного числа десятичных разрядов
INSERT	Вставляет подстроку в строку в заданной позиции для определенного числа символов
INSTR	Возвращает позицию первого вхождения строки в другую строку
LCASE	Преобразует строку в нижний регистр
LEFT	Извлекает подстроку из строки (начиная с левого)
LENGTH	Возвращает длину указанной строки (в байтах)
LOCATE	Возвращает позицию первого вхождения подстроки в строку
LOWER	Преобразует строку в нижний регистр
LPAD	Возвращает строку, которая заполнена с заданной строкой до определенной длины
LTRIM	Удаление начальных пробелов из строки
MID	Извлекает подстроку из строки (начиная с любой позиции)
POSITION	Возвращает позицию первого вхождения подстроки в строку
REPEAT	Повторяет строку указанное количество раз
REPLACE	Заменяет все вхождения указанной строки
REVERSE	Изменяет строку и возвращает результат
RIGHT	Извлекает подстроку из строки (начиная справа)
RPAD	Возвращает строку, которая имеет правую прокладку с указанной строкой до определенной длины
RTRIM	Удаляет замыкающие пробелы из строки
SPACE	Возвращает строку с заданным количеством пробелов
STRCMP	Проверяет, совпадают ли две строки
SUBSTR	Извлекает подстроку из строки (начиная с любой позиции)
SUBSTRING	Извлекает подстроку из строки (начиная с любой позиции)
SUBSTRING_INDEX	Возвращает подстроку строки перед числом вхождений разделителя
TRIM	Удаление начальных и конечных пробелов из строки
UCASE	Преобразует строку в верхний регистр
UPPER	Преобразует строку в верхний регистр

POW	
POWER	
RADIANS	Преобразует значение в градусах в радианы
RAND	Возвращает случайное число или случайное число в пределах диапазона
ROUND	Возвращает число, округленное до определенного числа десятичных разрядов
SIGN	Возвращает значение, указывающее знак числа
SIN	Возвращает синус числа
SQRT	Возвращает квадратный корень числа
UM	Возвращает суммированное значение выражения
AN	Возвращает тангенс числа
UNCATE	Возвращает число, усеченное до определенного числа десятичных разрядов

MySQL Дата функции

ADDDATE	Возвращает дату после добавления определенного интервала времени/даты
ADDTIME	Возвращает время/DateTime после добавления определенного временного интервала
CURDATE	Возвращает текущую дату
CURRENT_DATE	Возвращает текущую дату
CURRENT_TIME	Возвращает текущее время
CURRENT_TIMESTAMP	Возвращает текущую дату и время
CURTIME	Возвращает текущее время
DATE	Извлекает значение даты из выражения Date или DateTime
DATEDIFF	Возвращает разницу в днях между двумя значениями даты
DATE_ADD	Возвращает дату после добавления определенного интервала времени/даты
DATE_FORMAT	Форматирует дату, указанную в маске форматирования
DATE_SUB	Возвращает дату после вычитания определенного интервала времени/даты
DAY	Возвращает часть дня значения даты
DAYNAME	Возвращает имя дня недели для даты
DAYOFMONTH	Возвращает часть дня значения даты
DAYOFWEEK	Возвращает индекс дня недели для значения даты

DAYOFYEAR	Возвращает день года для значения даты
EXTRACT	Извлечение деталей из даты
FROM_DAYS	Возвращает значение даты из числового представления дня
HOUR	Возвращает часовую часть значения даты
LAST_DAY	Возвращает последний день месяца для заданной даты
LOCALTIME	Возвращает текущую дату и время
LOCALTIMESTAMP	Возвращает текущую дату и время
MAKEDATE	Возвращает дату для определенного года и дня года значение
MAKETIME	Возвращает время для определенного часа, минуты, второй комбинации
MICROSECOND	Возвращает микросекундную часть значения даты
MINUTE	Возвращает минутную часть значения даты
MONTH	Returns the month portion of a date value
MONTHNAME	Возвращает полное имя месяца для даты
NOW	Возвращает текущую дату и время
PERIOD_ADD	Занимает период и добавляет указанное количество месяцев к нему
PERIOD_DIFF	Возвращает разницу в месяцах между двумя периодами
QUARTER	Возвращает четвертую часть значения даты
SECOND	Возвращает вторую часть значения даты

SEC_TO_TIME	Преобразует числовые секунды в значение времени
STR_TO_DATE	Принимает строку и возвращает дату, указанную маской форматирования
SUBDATE	Возвращает дату, после которой был вычтен определенный интервал времени/даты
SUBTIME	Возвращает значение времени/DateTime после вычитания определенного временного интервала
SYSDATE	Возвращает текущую дату и время
TIME	Извлекает значение времени из выражения Time/DateTime
TIME_FORMAT	Форматирует время, указанное в маске форматирования
TIME_TO_SEC	Преобразует значение времени в числовые секунды
TIMEDIFF	Возвращает разницу между двумя значениями времени/DateTime
TIMESTAMP	Преобразует выражение в значение DateTime и при указании добавляет дополнительный временной интервал к значению
TO_DAYS	Преобразует дату в числовые дни
WEEK	Возвращает часть недели значения даты
WEEKDAY	Возвращает индекс дня недели для значения даты
WEEKOFYEAR	Возвращает неделю года для значения даты
YEAR	Возвращает часть года значения даты
YEARWEEK	Возвращает год и неделю для значения даты

BIN	Преобразует значение в двоичную строку
BINARY	Преобразует последующее число в двоичное число
CASE	Проверяет значение на соответствие списку условий и возвращает значение при обнаружении первого условия
CAST	Позволяет объявить условие и возвращает значение при обнаружении первого условия
COALESCE	Преобразует значение из одного типа данных в другой тип данных
CONNECTION_ID	Возвращает первое выражение, не являющееся null, в списке
CONV	Возвращает уникальный идентификатор подключения для текущего подключения
CONVERT	Преобразует число из одной числовой базы в другую
CURRENT_USER	Преобразует значение из одного типа данных в другой или один набор символов в другой
DATABASE	Возвращает имя пользователя и имя хоста для учетной записи MySQL, используемой сервером для проверки подлинности текущего клиента
IF	Возвращает имя базы данных по умолчанию
IFNULL	Возвращает одно значение, если условие имеет значение true, или другое значение, если условие имеет значение false
ISNULL	Позволяет возвращать альтернативное значение, если выражение имеет значение null
LAST_INSERT_ID	Проверяет, является ли выражение null
NULLIF	Возвращает первое значение auto_инкремент, которое было задано последней инструкцией INSERT или Update
SESSION_USER	Сравнивает два выражения
SYSTEM_USER	Возвращает имя пользователя и имя хоста для текущего пользователя MySQL
USER	Возвращает имя пользователя и имя хоста для текущего пользователя MySQL
VERSION	Возвращает имя пользователя и имя хоста для текущего пользователя MySQL
	Возвращает версию базы данных MySQL

Расширенные функции MySQL

7.8. Простые запросы

Оператор SELECT

Оператор SELECT – один из наиболее важных и используемых операторов SQL. Он позволяет производить выборки данных из БД и преобразовывать к нужному виду полученные результаты.

Оператор SELECT – полностью абстрагирован от вопросов представления данных, всё внимание при его применении сконцентрировано на проблемах доступа к данным.

Запросы на чтение данных. Оператор SELECT

Синтаксис оператора SELECT

SELECT [**ALL** | **DISTINCT**] список_возвращаемых_столбцов|*
FROM список_имен_таблиц
[**WHERE** условие_поиска]
[**GROUP BY** список_имен_столбцов]
[**HAVING** условие_поиска]
[**ORDER BY** имя_столбца [**ASC** | **DESC**],…]

Примечание: в квадратных скобках указаны предложения, которые могут отсутствовать в операторе SELECT.

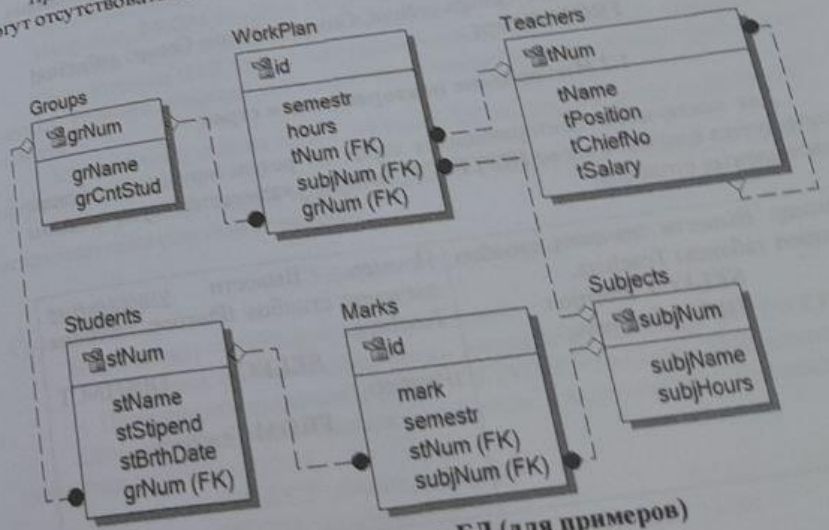


Рисунок. Схема БД (для примеров)

1. Цель запроса. Предложение SELECT

Предложение SELECT содержит список возвращаемых столбцов, разделенных символом «запятая» (,).

1.1 Способы указания выводимых столбцов

– **вывод значений определенных столбцов** одной из таблицы, указанной в предложении FROM

Пример. Получить список студентов и размер их стипендий.

SELECT stName, stStipend
FROM Students;

– для **вывода всех столбцов** таблицы, указанной в предложении FROM, можно перечислить все их названия или воспользоваться символом «звездочка»

Пример. Вывести все столбцы таблицы Groups.

SELECT grNum, grName, grCntStud
FROM Groups;

или

SELECT *
FROM Groups;

– **уточнение имен** столбцов путем указания полного имени столбца: **имя_таблицы.имя_столбца**.

Пример:

SELECT Groups.grNum, Groups.grName, Groups.grCntStud
FROM Groups ;

1.2 Исключение повторяющихся строк

Для исключения повторяющихся строк из результирующей таблицы используется ключевое слово **DISTINCT**, которое указывается перед списком возвращаемых столбцов.

имер. Вывести значения столбца ition таблицы Teachers. **SELECT** tPosition **FROM** Teachers;	Пример. Вывести уникальные значения столбца tPosition таблицы Teachers. **SELECT** DISTINCT tPosition **FROM** Teachers;

1.3 Использование вычисляемых выражений

Пример. Вывести фамилии студентов, размер их стипендий и в $.
SELECT stName, stStipend, stStipend / 9850 **FROM** Students ;

1.4 Переопределение имен результирующих столбцов

Для переопределения имени результирующего столбца (создания его синонима) используется ключевое слово **AS**.

1.5 Включение текста в результат запроса

В предложении SELECT кроме имен столбцов и выражений с ними можно указывать константы (и константные выражения).

Пример. Вывести фамилии студентов и размер их стипендий, оформив результат предложениями на русском языке.

SELECT 'Студент', stName, **'получает стипендию'**, stStipend
FROM Students ;

2. Используемые таблицы. Предложение FROM

Предложение FROM содержит список имен таблиц, разделенных символом «запятая» (,).

Например, **FROM Students, Groups**.

Можно указывать **синонимы (псевдонимы) имен таблиц**.

Например, **FROM** Students **st**, Groups **gr**

3. Отбор строк. Предложение WHERE

Предложение WHERE состоит из ключевого слова **WHERE**, за которым следует **условие поиска**, определяющее, какие именно строки требуется выбрать.

Если условие поиска имеет значение TRUE, строка будет включена в результат запроса.

Если условие поиска имеет значение FALSE или NULL, то строка исключается из результата запроса.

3.1 Условия отбора строк

Сравнение

Выражение1 =|<>|<|>|<=|>= Выражение2

Проверка на принадлежность диапазону значений (BETWEEN).

Проверяемое выражение **[NOT] BETWEEN** минимум **AND** максимум

Пример. Получить список студентов, получающих стипендию в диапазоне от 650 до 1100 $.

SELECT stName, stStipend
FROM Students
WHERE stStipend **BETWEEN** 650 **AND** 1100;

Проверка на принадлежность множеству (IN)

проверяемое_выражение **[NOT] IN** (*набор_констант*)

Пример. Получить список студентов, получающих стипендию 650, или 730, или 900 $.

SELECT stName, stStipend
FROM Students
WHERE stStipend **IN** (650, 730, 900);

Проверка на соответствие шаблону (LIKE)

имя_столбца **[NOT] LIKE** *шаблон* [ESCAPE *символ_пропуска*],

где *шаблон* – это строка, в которую может входить один или более подстановочных знаков. подстановочные знаки

% – совпадает с любой последовательностью из нуля или более символов

Пример. Получить сведения о студентах, чья фамилия начинается с «Соби».

SELECT *
FROM Students
WHERE stName **LIKE 'Соби%'**;

_ (символ подчеркивания) – совпадает с любым отдельным символом.

Пример. Получить сведения о студентах, чье имя «Наталья» или «Наталия».

SELECT *
FROM Students
WHERE stName **LIKE '%Натал_я'**;

символ пропуска используется для проверки наличия в строках символов, использующихся в качестве подстановочных знаков (%, _).

Пример. Получить сведения из таблицы "Data", где в поле результат содержится фрагмент текста "менее 50%".

SELECT *
FROM Data
WHERE Result **LIKE** '%менее 50**$%** %' **ESCAPE $**;

Проверка на равенство значению NULL (IS NULL)

имя_столбца **IS [NOT] NULL**

Пример. Получить сведения о студентах, получающих стипендию.

SELECT stName, stNum, stStipend
FROM Students
WHERE stStipend **IS NOT NULL**;

Составные условия поиска (AND, OR и NOT)

WHERE [NOT] условие_поиска **[AND|OR]** [NOT] условие_поиска ...

Пример. Получить сведения о студентах, которые учатся в группе с кодом «1» и получают стипендию.

SELECT *
FROM Students
WHERE (grNum = 1) **AND** (stStipend IS NOT NULL);

7.9. Агрегатные функции

Агрегатная функция принимает в качестве аргумента какой-либо столбец данных целиком, а возвращает одно значение, которое определенным образом подытоживает этот столбец.

- **SUM**(выражение | [DISTINCT] имя_столбца) – сумма [различных] числовых значений
- **AVG**(выражение | [DISTINCT] имя_столбца) – средняя величина [различных] числовых значений
- **MIN**(выражение | имя_столбца) – наименьшее среди всех значений
- **MAX**(выражение | имя_столбца) – наибольшее среди всех значений
- **COUNT**([DISTINCT] имя_столбца) – подсчитывает количество значений, содержащихся в столбце
- **COUNT**(*) – подсчитывает количество строк в таблице результатов запроса

Примечание: агрегатные функции нельзя применять в предложении WHERE.

Пример 1. Найти суммарное, среднее, минимальное и максимальное значение стипендии студентов.

SELECT SUM(stStipend) AS Sm, AVG(stStipend) AS Av, MIN(stStipend) AS Mn, MAX(stStipend) AS Mx **FROM** Students;

Пример 2. Найти количество студентов, получающих стипендию.

SELECT COUNT(*) AS Cnt
FROM Students
WHERE stStipend > 0;

Контрольные вопросы

1. История развития SQL. Стандарты языка SQL.
2. Назначение языка SQL. Составные части языка SQL.
3. Диалекты языка SQL. Режимы работы с базой данных.
4. Структура (группы операторов) языка SQL.
5. SQL операторы определения данных DDL
6. SQL операторы манипулирования данными (DML)
7. SQL операторы защиты и управления данными (DCL)
8. SQL команды управления транзакциями (TCL)
9. Типы данных языка SQL (MySQL)
10. Встроенные функции. MySQL строковые функции.
11. Встроенные функции. Числовые функции MySQL.

12. Встроенные функции. Функции работы с датой MySQL.
13. Встроенные функции. Расширенные функции MySQL.
14. Объясните формат инструкции SELECT.
15. Запросы на чтение данных. Оператор SELECT.
16. Оператор SELECT: предложение FROM и WHERE. (У
строк: Сравнение, BETWEEN, IN, Проверка на соответствие шаблону (LIKE
и др.)
17. Применение агрегатных функций в операторе SELECT

8. ОПРЕДЕЛЕНИЯ ДАННЫХ В SQL

8.1. Оператор создания таблиц CREATE TABLE

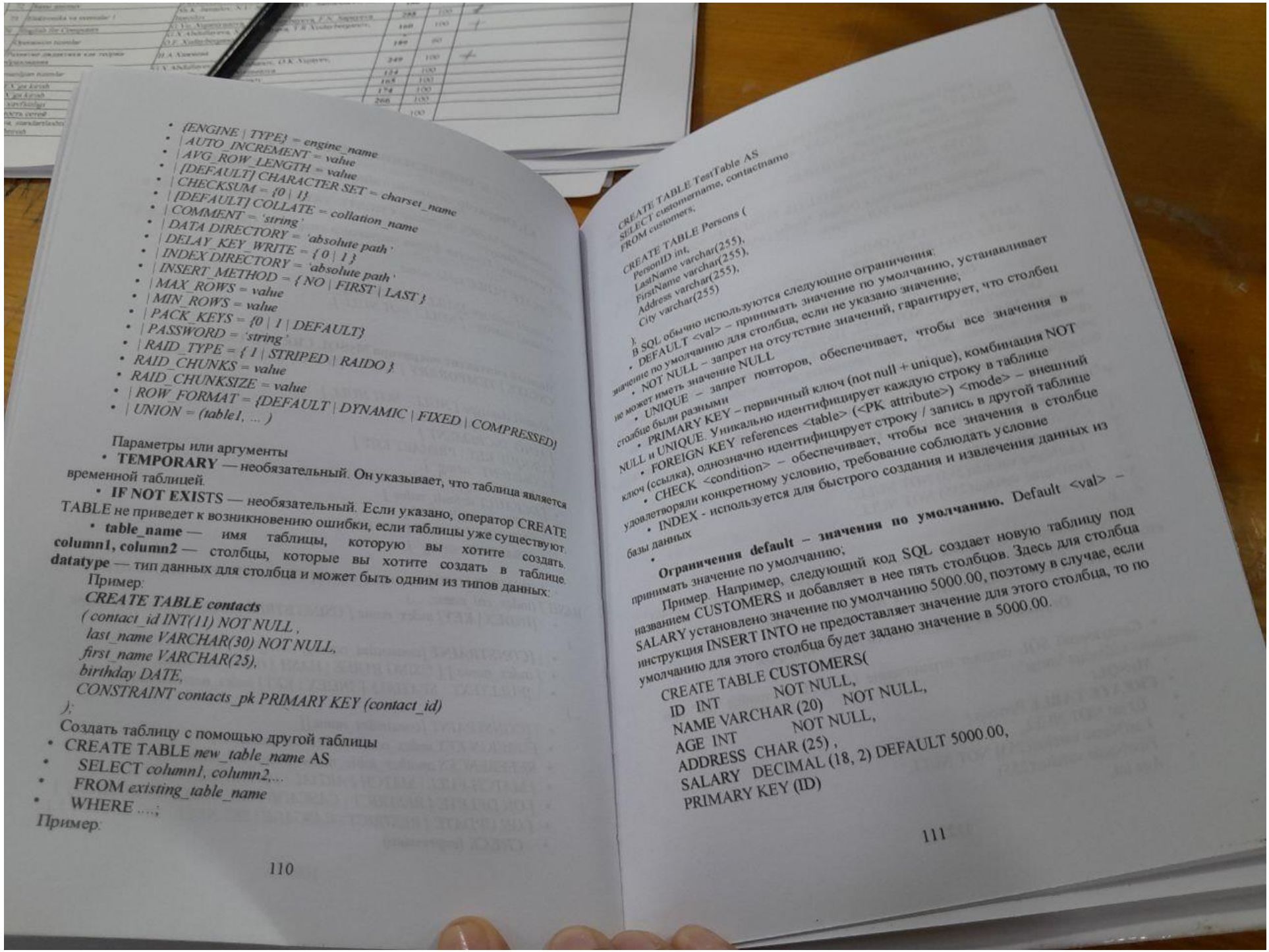
Описание: MySQL оператор **CREATE TABLE** позволяет создавать и определять таблицу.

Синтаксис: Простая форма синтаксиса оператора CREATE TABLE в MySQL:

```
CREATE TABLE table_name
( column1 datatype [NULL | NOT NULL],
column2 datatype [ NULL | NOT NULL ],
...
);
```

Полный синтаксис оператора MySQL CREATE TABLE:

```
CREATE [ TEMPORARY ] TABLE [IF NOT EXISTS] table_name
(
column1 datatype [ NULL | NOT NULL ]
[DEFAULT default_value ]
[AUTO_INCREMENT ]
[UNIQUE KEY | PRIMARY KEY ]
[COMMENT 'string' ],
column2 datatype [ NULL | NOT NULL ]
[DEFAULT default_value ]
[AUTO_INCREMENT ]
[UNIQUE KEY | PRIMARY KEY ]
[COMMENT 'string' ],
...
• \ [CONSTRAINT [constraint_name]] PRIMARY KEY [ USING BTREE \
HASH ] (index_col_name, ...)
• \ [INDEX \ KEY] index_name [ USING BTREE \ HASH ] (index_col_name,
...)
• \ [CONSTRAINT [constraint_name]] UNIQUE [ INDEX \ KEY ]
[ index_name ] [ USING BTREE \ HASH ] (index_col_name, ...)
• \ {FULLTEXT \ SPATIAL} [ INDEX \ KEY] index_name (index_col_name,
...)
• \ [CONSTRAINT [constraint_name]]
• FOREIGN KEY index_name (index_col_name, ...)
• REFERENCES another_table_name (index_col_name, ...)
• [ MATCH FULL \ MATCH PARTIAL \ MATCH SIMPLE ]
• [ ON DELETE { RESTRICT \ CASCADE \ SET NULL \ NO ACTION } ]
• [ ON UPDATE { RESTRICT \ CASCADE \ SET NULL \ NO ACTION } ]
• \ CHECK (expression)
```



• {ENGINE | TYPE} = engine_name
• | AUTO_INCREMENT = value
• | AVG_ROW_LENGTH = value
• | [DEFAULT] CHARACTER SET = charset_name
• | CHECKSUM = {0 | 1}
• | [DEFAULT] COLLATE = collation_name
• | COMMENT = 'string'
• | DATA DIRECTORY = 'absolute path'
• | DELAY_KEY_WRITE = { 0 | 1 }
• | INDEX DIRECTORY = 'absolute path'
• | INSERT_METHOD = { NO | FIRST | LAST }
• | MAX_ROWS = value
• | MIN_ROWS = value
• | PACK_KEYS = {0 | 1 | DEFAULT}
• | PASSWORD = 'string'
• | RAID_TYPE = { 1 | STRIPED | RAID0 }
• RAID_CHUNKS = value
• RAID_CHUNKSIZE = value
• | ROW_FORMAT = {DEFAULT | DYNAMIC | FIXED | COMPRESSED}
• | UNION = (table1, ...)

Параметры или аргументы

• **TEMPORARY** — необязательный. Он указывает, что таблица является временной таблицей.

• **IF NOT EXISTS** — необязательный. Если указано, оператор CREATE TABLE не приведет к возникновению ошибки, если таблицы уже существуют.

• **table_name** — имя таблицы, которую вы хотите создать.

column1, column2 — столбцы, которые вы хотите создать в таблице.

datatype — тип данных для столбца и может быть одним из типов данных:

Пример:

```
CREATE TABLE contacts
( contact_id INT(11) NOT NULL ,
 last_name VARCHAR(30) NOT NULL,
 first_name VARCHAR(25),
 birthday DATE,
 CONSTRAINT contacts_pk PRIMARY KEY (contact_id)
);
```

Создать таблицу с помощью другой таблицы

```
• CREATE TABLE new_table_name AS
•   SELECT column1, column2,...
•   FROM existing_table_name
•   WHERE ....;
```

Пример:


```
CREATE TABLE TestTable AS
SELECT customername, contactname
FROM customers;

CREATE TABLE Persons (
 PersonID int,
 LastName varchar(255),
 FirstName varchar(255),
 Address varchar(255),
 City varchar(255)
);
```

В SQL обычно используются следующие ограничения:

значение по умолчанию для столбца, если не указано значение;

• DEFAULT <val> – принимать значение по умолчанию, устанавливает

• NOT NULL – запрет на отсутствие значений, гарантирует, что столбец не может иметь значение NULL

• UNIQUE – запрет повторов, обеспечивает, чтобы все значения в столбце были разными

• PRIMARY KEY – первичный ключ (not null + unique), комбинация NOT NULL и UNIQUE. Уникально идентифицирует каждую строку в таблице

• FOREIGN KEY references <table> (<PK attribute>) <mode> – внешний ключ (ссылка), однозначно идентифицирует строку / запись в другой таблице

• CHECK <condition> – обеспечивает, чтобы все значения в столбце удовлетворяли конкретному условию, требование соблюдать условие

• INDEX - используется для быстрого создания и извлечения данных из базы данных

•

Ограничения default – значения по умолчанию. Default <val> – принимать значение по умолчанию;

Пример. Например, следующий код SQL создает новую таблицу под названием CUSTOMERS и добавляет в нее пять столбцов. Здесь для столбца SALARY установлено значение по умолчанию 5000.00, поэтому в случае, если инструкция INSERT INTO не предоставляет значение для этого столбца, то по умолчанию для этого столбца будет задано значение в 5000.00.

```
CREATE TABLE CUSTOMERS(
ID   INT              NOT NULL,
NAME VARCHAR (20)     NOT NULL,
AGE  INT              NOT NULL,
ADDRESS  CHAR (25) ,
SALARY   DECIMAL (18, 2) DEFAULT 5000.00,
PRIMARY KEY (ID)
```

Если таблица CUSTOMERS уже создана, то для добавления ограничения DEFAULT для столбца SALARY вы должны сделать запрос, который приведен в блоке кода ниже.

Изменение таблицы CUSTOMERS

```
ALTER TABLE customers
MODIFY SALARY DECIMAL (18, 2) DEFAULT 5000.00;
```

Удаление ограничения Default. Чтобы удалить ограничение DEFAULT, используйте следующий SQL-запрос

```
ALTER TABLE CUSTOMERS
ALTER COLUMN SALARY DROP DEFAULT;
```

Ограничения NOT NULL–не пустое значения

Ограничение SQL NOT NULL. По умолчанию столбец может содержать значения NULL. Ограничение NOT NULL обеспечивает, чтобы столбец не принимал значения NULL. Это применяет поле, чтобы всегда содержать значение, что означает, что нельзя вставить новую запись или обновить запись без добавления значения в это поле.

Следующий SQL гарантирует, что столбцы "ID", "Фамилия -"LastName и "имя-FirstName " не будут принимать значения NULL:

Пример. Изменение таблицы **PERSONS:**

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255) NOT NULL,
    Age int
);
```

Если таблица уже создана, можно добавить ограничение NOT NULL для столбец с инструкцией ALTER TABLE.

ALTER TABLE Persons **MODIFY** Age int NOT NULL.

Ограничение уникальности SQL – UNIQUE

- Следующий SQL создает ограничение UNIQUE в столбце "ID" при ании таблицы "лица":
- **MySQL:**
- **CREATE TABLE** Persons (
  ID int NOT NULL,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Age int,

- UNIQUE (ID)
-);

Чтобы назвать ограничение UNIQUE и определить уникальное ограничение для нескольких столбцов, используйте следующий синтаксис SQL:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CONSTRAINT UC_Person UNIQUE (ID,LastName)
);
```

Ограничение уникальности SQL при изменении таблицы:

```
ALTER TABLE Persons ADD UNIQUE (ID);
```

Чтобы назвать ограничение UNIQUE и определить уникальное ограничение для нескольких столбцов, используйте следующий синтаксис SQL:

```
ALTER TABLE Persons
ADD CONSTRAINT UC_Person UNIQUE (ID,LastName);
```

Ограничение первичного ключа SQL- PRIMARY KEY.

Следующий SQL создает первичный ключ в столбце "ID" при создании таблицы "персоны":

- **CREATE TABLE** Persons (
- ID int NOT NULL,
- LastName varchar(255) NOT NULL,
- FirstName varchar(255),
- Age int,
- PRIMARY KEY (ID)
-);

Чтобы разрешить именование ограничения первичного ключа, а также для определения ограничения первичного ключа для нескольких столбцов, используйте следующий синтаксис SQL:

- **CREATE TABLE** Persons (
- ID int NOT NULL,
- LastName varchar(255) NOT NULL,
- FirstName varchar(255),
- Age int,
- CONSTRAINT PK_Person PRIMARY KEY (ID,LastName)
-);

8.2. ALTER TABLE оператор MySQL. Изменения структуры таблицы

MySQL оператор ALTER TABLE используется для добавления, изменения или удаления столбцов в таблице. Оператор MySQL ALTER TABLE также используется для переименования таблицы.

1.Добавить столбец в таблицу

Синтаксис добавления столбца в таблицу MySQL (с использованием оператора ALTER TABLE):

- **ALTER TABLE** table_name
- **ADD** new_column_name column_definition
- [**FIRST** | **AFTER** column_name];
- *table_name* — имя таблицы для изменения.
- *new_column_name* — имя нового столбца для добавления в таблицу.
- *column_definition* — тип данных и определение столбца (NULL или NOT NULL и т. д.).
- *FIRST | AFTER column_name* — необязательный. Он сообщает MySQL, где в таблице создается столбец. Если этот параметр не указан, то новый столбец будет добавлен в конец таблицы.

Пример:

```
ALTER TABLE contacts
ADD last_name varchar(40) NOT NULL
AFTER contact_id;
```

Этот MySQL пример ALTER TABLE добавит столбец с именем last_name в таблицу contacts. Он будет создан как столбец NOT NULL и появится в таблице после поля contact_id.

Пример2:

```
ALTER TABLE contacts
ADD last_name varchar(40) NOT NULL
AFTER contact_id,
ADD first_name varchar(35) NULL
AFTER last_name;
```

Этот пример ALTER TABLE добавит в таблицу contacts два столбца st_name и first_name.

Поле last_name будет создано как столбец varchar (40) NOT NULL и тся в таблице contacts после столбца contact_id. ец first_name будет создан как столбец NULL varchar (35) и появится в е после столбца last_name.

Изменить столбец в таблице. Синтаксис для изменения столбца в MySQL (с использованием оператора ALTER TABLE):

```
ALTER TABLE table_name
MODIFY column_name column_definition
[FIRST | AFTER column_name ];
```


Пример:

```
ALTER TABLE contacts
MODIFY last_name varchar(50) NULL;
```

Удаление столбца из таблицы. Синтаксис для удаления столбца из таблицы в MySQL (с использованием оператора ALTER TABLE):

Например:

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

Пример:

```
ALTER TABLE contacts
DROP COLUMN contact_type;
```

Переименование столбца в таблице. Синтаксис для переименования столбца в таблице MySQL (с использованием оператора ALTER TABLE):

- ALTER TABLE table_name
- CHANGE COLUMN old_name new_name
- column_definition
- [*FIRST* | AFTER column_name];
- Пример:

```
ALTER TABLE contacts
CHANGE COLUMN contact_type ctype
varchar(20) NOT NULL;
```

Переименовать таблицу. Синтаксис для переименования таблицы в MySQL:

- ALTER TABLE table_name
- RENAME TO new_table_name;

Пример. Этот пример ALTER TABLE переименует таблицу contacts в people.

```
ALTER TABLE contacts
RENAME TO people;
```

8.3. Обновления данных, оператор UPDATE

Описание

MySQL оператор UPDATE используется для обновления существующих записей в таблице в базе данных MySQL. Существует три синтаксиса для оператора UPDATE в зависимости от типа обновления, которое вы хотите выполнить.

Синтаксис. Простая форма синтаксиса для оператора UPDATE при обновлении одной таблицы в MySQL:

```
UPDATE table
SET column1 = expression1,
```

column2 = expression2,
...
[WHERE conditions];

Пример обновление одного столбца. Рассмотрим очень простой пример MySQL запроса UPDATE:

```
UPDATE customers
SET last_name = 'Ford'
WHERE customer_id = 500;
```

Рассмотрим пример MySQL UPDATE, где вы можете обновить более одного столбца с помощью одного оператора UPDATE.

```
UPDATE customers
SET state = 'Nevada',  customer_rep = 23
WHERE customer_id > 200;
```

Пример обновления таблицы данными из другой таблицы:

```
UPDATE customers
SET city = (SELECT city
    FROM suppliers
    WHERE suppliers.suppliemame = customers.customername)
WHERE customerid > 5000;
```

8.4. Вставка данных. оператор INSERT INTO

Описание. MySQL оператор INSERT используется для вставки одной записи или нескольких записей в таблицу в MySQL.

Синтаксис простой формы оператора INSERT для вставки одной записи с использованием ключевого слова VALUES в MySQL:

```
INSERT INTO table
(column1, column2, ... )
VALUES
(expression1, expression2, ... ),
(expression1, expression2, ... ),
...;
```

Синтаксис простой формы оператора INSERT для вставки нескольких писей с использованием подзапроса в MySQL:

```
INSERT INTO table
(column1, column2, ... )
SELECT expression1, expression2, ...
FROM source_table[WHERE conditions];
```

Пример использования ключевого слова VALUES. Самый простой создать MySQL запрос INSERT для отображения значений с помощью ого слова VALUES.

пример:


```
1  INSERT INTO suppliers
2  (supplier_id, supplier_name)
3  VALUES
4  (100, 'Acer');
```

Этот MySQL пример INSERT приведет к тому, что одна запись будет вставлена в таблицу suppliers. Новая запись будет иметь supplier_id 100 и supplier_name 'Acer'.

Пример с использованием подзапроса. Вы также можете создавать более сложные MySQL операторы INSERT, используя подзапросы.Например:

```
INSERT INTO suppliers
(supplier_id, supplier_name)
SELECT account_no, name
FROM customers
WHERE customer_id < 300,
```

Контрольные вопросы

1. Опишите CREATE TABLE - оператор создания таблиц в SQL. Приведите пример по создания таблиц.
2. Опишите оператор ALTER TABLE - изменения структуры таблицы. Какие операции по изменения структуры таблицы выполняется оператором? Приведите примеры.
3. Опишите оператор UPDATE - обновления данных. Какие операции обновления выполняется оператором? Приведите примеры.
4. Опишите оператор INSERT INTO - вставка данных. Какие операции вставки выполняется оператором? Приведите примеры.

9. ВЛОЖЕННЫЕ И СЛОЖНЫЕ ЗАПРОСЫ SQL

9.1. Сортировка результатов запроса. Предложение ORDER BY

ORDER BY имя_столбца [ASC | DESC]

где, ASC – возрастающий, DESC – убывающий порядок сортировки.

Пример. Вывести список фамилий студентов, учащихся в группе КИ-125 в обратном алфавитном порядке.

```
SELECT stName
FROM Students, Groups
WHERE Students.stNum = Groups.grNum AND
Groups.grName = 'КИ-125'
ORDER BY stName DESC
```

Использование фразы GROUP BY позволяет сгруппировать строки в группы, имеющие одинаковые значения указанного поля:

grName
КИ-121
ПИ-111
КИ-101
КИ-121

=

ORDER BY grName
КИ-101
КИ-121
КИ-121
ПИ-111

=

GROUP BY grName
КИ-101
КИ-121
ПИ-111

К группам, полученным после применения GROUP BY, можно применить любую из стандартных агрегатных функций.

Пример 1. Получить список студентов и их средний балл.

```
SELECT stName, AVG(mark) AS AvgMark
FROM Students, Marks
WHERE Students.stNum = Marks.stNum
GROUP BY stName
```

Примечание. В списке отбираемых полей оператора SELECT, содержащего раздел GROUP BY, можно включать только агрегатные функции и поля, которые входят в условие группировки.

Несколько столбцов группировки

Пример. Получить список студентов и их средний балл за каждый семестр.

```
SELECT stName, semestr, AVG(mark) AS AvgMark
FROM Students, Marks
WHERE Students.stNum = Marks.stNum
GROUP BY stName, semester
```


Значения NULL в столбцах группировки

Строки, имеющие значение NULL в одинаковых столбцах группировки и идентичные значения во всех остальных столбцах группировки, помещаются в одну группу.

Условия поиска групп. Предложение HAVING

Предложение HAVING, используемое совместно с GROUP BY, позволяет исключить из результата группы, неудовлетворяющие условию (так же, как WHERE позволяет исключить строки).

Пример 1. Получить список групп специальности КИ, в которых число студентов меньше 15.

```
SELECT grName, COUNT(*) AS CntStudents
FROM Students, Groups
WHERE Students.gtNum = Groups.grNum AND
Groups.grName LIKE 'КИ%'
GROUP BY grName
HAVING COUNT(*) < 15
```

9.2. *Вложенные запросы*

Вложенным запросом (подзапросом) называется запрос, содержащийся в предложении WHERE или HAVING другого оператора SQL.

Пример 1. Получить список предметов, по которым была получена оценка <4.

```
SELECT subjName
FROM Subjects
WHERE subjNum IN ( SELECT subjNum
                   FROM Marks
                   WHERE mark < 4)
```

Коррелируемым подзапросом называется подзапрос, который содержит ссылку на столбцы таблицы внешнего запроса.

Пример 2. Вывести список студентов, средний балл которых выше 4,5.

```
SELECT stName
FROM Students
WHERE (SELECT AVG(mark) FROM Marks
       WHERE Marks.stNum = Students.stNum) > 4.5
```

Особенности вложенных запросов:

- вложенный запрос всегда заключается в круглые скобки;
- таблица результатов вложенного запроса всегда состоит из одного столбца;
- во вложенный запрос не может входить предложение ORDER BY.

9.3. Квантор существования EXISTS

В языке SQL предикат с квантором существования представляется выражением вида:

[NOT] EXISTS (SELECT…FROM…WHERE…),

которое следует за фразой WHERE. Такое выражение считается истинным, если подзапрос возвращает непустое множество (существует хотя бы 1 строка, которую возвращает подзапрос). На практике подзапрос всегда будет коррелированным.

Пример. Получить список студентов, сдавших хотя бы один экзамен.

```
SELECT stName FROM Students
WHERE EXISTS ( SELECT * FROM Marks
WHERE Marks.stNum = Students.stNum);
```

9.4. Многократное сравнение ANY и ALL

Синтаксис многократного сравнения:

проверяемое_выражение = | <> | < | <= | > | >=

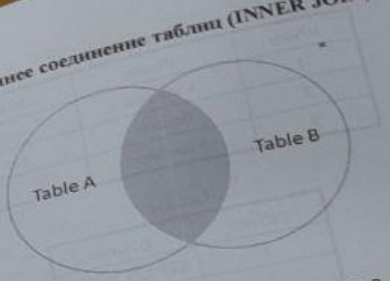
ANY | ALL вложенный_запрос

9.5. Квантор общности ALL в языке SQL

Пример. Получить список студентов, получающих стипендию большую, чем любой из студентов группы КИ-121.

```
SELECT *
FROM Students
WHERE stStipend > ALL (SELECT stStipend
                       FROM Students, Groups
                       WHERE Students.grNum = Groups.grNum
                           AND Groups.grName = 'КИ-121');
```


9.6. Внутреннее соединение таблиц (INNER JOIN)



INNER JOIN

Пример. Вывести список студентов, и названия групп, в которых они учатся.

```
SELECT stName, grName
FROM Students INNER JOIN Groups
        ON Students.grNum = Groups.grNum;
```

Если таблицы нужно соединить по равенству столбцов с одинаковыми именами, то вместо предложения ON используется предложение USING, в котором перечисляются названия соединяемых столбцов.

Пример.

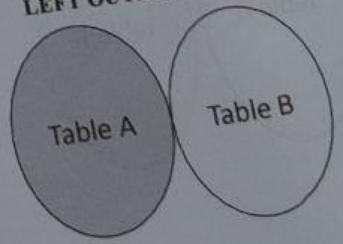
```
SELECT stName, grName
FROM Students INNER JOIN Groups
        USING (grNum);
```

9.7. Внешнее соединение таблиц (OUTER JOIN)

В SQL-92 поддерживается понятие **внешнего соединения** двух типов:

- левостороннее (LEFT OUTER JOIN, *=);
- правостороннее (RIGHT OUTER JOIN, =*).

LEFT OUTER JOIN



Students

stNum	stName	grNum
1	Собиров	1
2	Ваисов	1
3	Камолов	

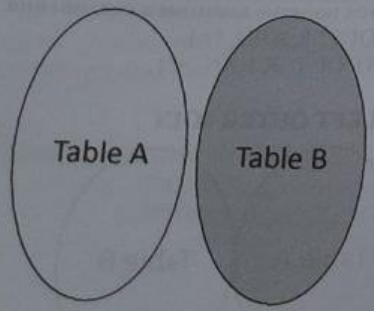
Groups

grNum	grName
1	КИ-121
2	ПИ-111

SELECT Students.stName, Groups.grName
FROM Students **LEFT OUTER JOIN** Groups **ON** Students.grNum = Groups.grNum
Результат:

stName	grName
Собиров	КИ-121
Ваисов	КИ-121
Камолов	

RIGHT OUTER JOIN



Students

stNum	stName	grNum
1	Собиров	1
2	Ваисов	1
3	Камолов	

Groups

grNum	grName
1	КИ-121
2	ПИ-111

SELECT Students.stName, Groups.grName
FROM Students **RIGHT OUTER JOIN** Groups **ON** Students.grNum = Groups.grNum
Результат:

stName	grName
Собиров	КИ-121
Ваисов	КИ-121
	ПИ-111

Контрольные вопросы

1. Сортировка результатов SELECT запроса и запросы с группировкой: предложение ORDER BY и GROUP BY.
2. Вложенные SELECT запросы.
3. Внутреннее и внешнее соединение таблиц (INNER JOIN и OUTER JOIN).

10.1. Хранимые процедуры MySQL

Хранимая процедура MySQL представляет собой подпрограмму, хранящуюся в базе данных. Она содержит имя, список параметров и операторы SQL. Все популярные системы управления базами данных поддерживают хранимые процедуры. Они были введены в MySQL 5, чтобы использовать их, необходимо иметь привилегию CREATE ROUTINE. Если двоичная регистрация допускается, эти инструкции могут также требовать привилегии SUPER.

Существует два вида подпрограмм: хранимые процедуры и функции, возвращающие значения, которые используются в других операторах SQL (например, pi()).

Основное отличие заключается в том, что функции могут использоваться, как любое другое выражение в операторах SQL, а хранимые процедуры должны вызываться с помощью оператора CALL.

10.2. В чем преимущество хранимых процедур?

Хранимые процедуры работают быстро. Преимущество сервера **MySQL** заключается в том, что он использует кэширование, а также заранее заданные операторы. Основной прирост скорости дает сокращение сетевого трафика. Если есть повторяющиеся задачи, которые требуют проверки, обработки циклов, нескольких операторов, и при этом не требуют взаимодействия с пользователем, это можно реализовать с помощью одного вызова процедуры, которая хранится на сервере;

MySQL хранимые процедуры являются универсальными. При написании хранимой процедуры на **SQL** она будет работать на любой платформе, которая использует **MySQL**. В этом преимущество **SQL** над другими языками, такими как **Java**, **C** или **PHP**;

Исходный код хранимых процедур всегда доступен в базе данных. Это эффективная практика связать данные с процессами, которые их обрабатывают.

10.3. Оператор создания процедур CREATE PROCEDURE.

Синтаксис

По умолчанию процедура связана с базой данных, используемой в ный момент. Чтобы связать процедуру с конкретной базой данных, жите ее создании хранимой процедуры: *базы_данных.имя_хранимой_процедуры*. Полный синтаксис:


```
CREATE [DEFINER = { user | CURRENT_USER }]
PROCEDURE имя_процедуры ([параметры_процедуры[,...]])
[characteristic ...] тело_подпрограммы
параметры_процедуры: [ IN | OUT | INOUT ] имя_параметра TYPE;
```

Здесь type любой валидный тип данных MySQL

характеристики: COMMENT 'string'

```
| LANGUAGE SQL
| [NOT] DETERMINISTIC
| { CONTAINS SQL | NO SQL | READS SQL DATA
| MODIFIES SQL DATA }
| { SECURITY { DEFINER | INVOKER }
| SQL SECURITY { DEFINER | INVOKER }
тело_подпрограммы: Валидный оператор программы SQL
```

4. Что такое параметры процедур? В принципе это тоже самое, что и параметры функций, процедур в языках программирования. Параметры разделяют на следующие виды.

- **IN** параметры. Входящие. Это то, что мы отправляем вместе с SQL запросом. Эти параметры уходят в процедуру, процедура их обрабатывает при исполнении.
- **OUT** параметры. Исходящие. Это что-то вроде результата. Процедура отработала и через OUT параметры мы можем получить свой результат
- **INOUT** параметры. Входящие и Исходящие одновременно. Их можно использовать как IN или как OUT или одновременно, например, при создании счетчика.

Вызов процедуры. Воссоздание процедуры. Оператор **CALL** используется для вызова процедуры, которая хранится в базе данных. Синтаксис следующий:

- **CALL имя_процедуры([параметр[,...]]) CALL имя_процедуры[()]**

MySQL хранимые процедуры, которые не принимают аргументов, могут вызываться без скобок.

Поэтому **CALL job_data()** равносильно **CALL job_data**.

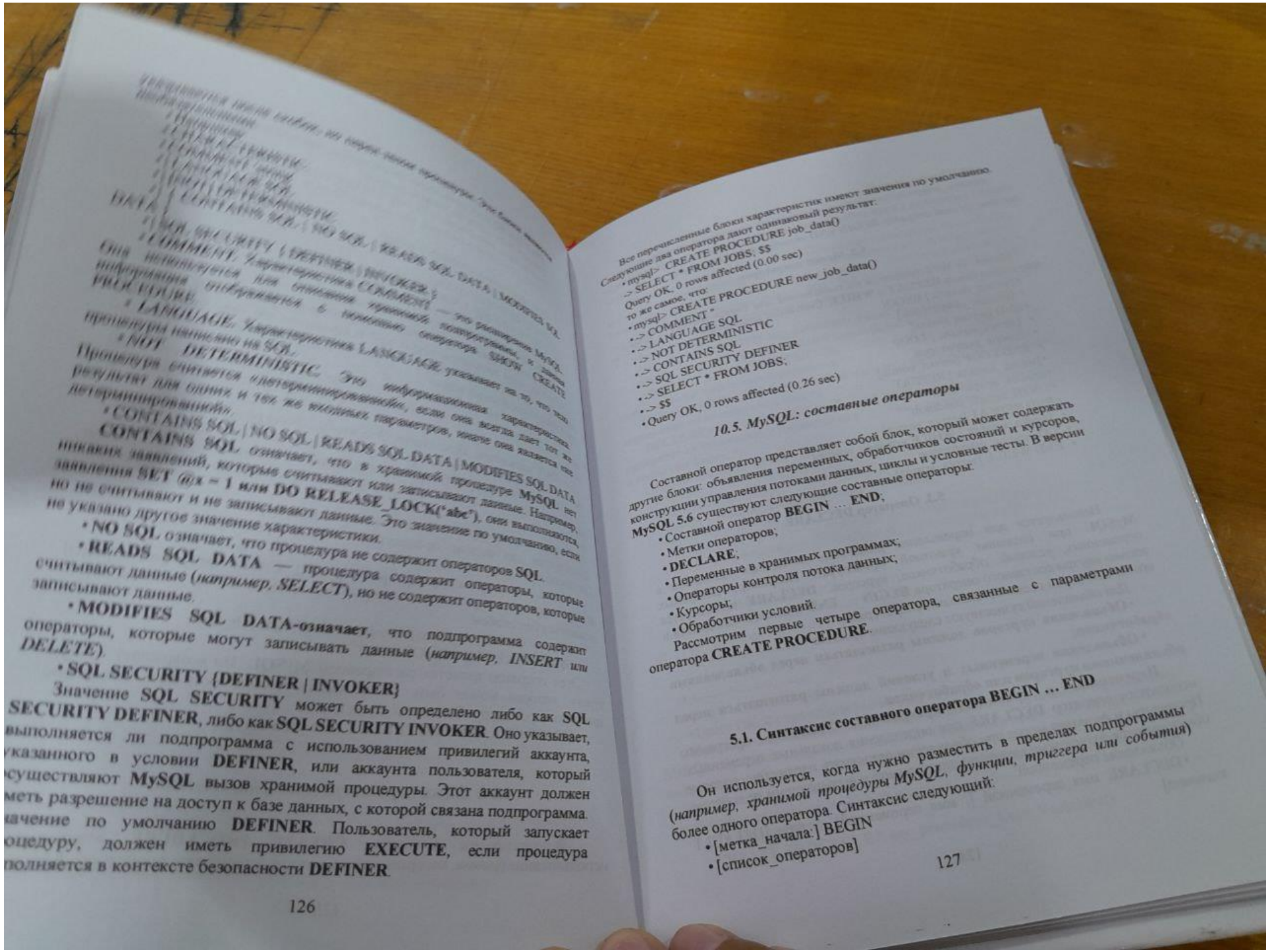
- **SHOW CREATE PROCEDURE**

Этот оператор является расширением **MySQL**. Он возвращает точную строку, которая может быть использована, чтобы воссоздать указанную хранимую процедуру. Синтаксис следующий:

- **SHOW CREATE PROCEDURE имя_процедуры**

10.4. Блоки характеристик characteristic

В синтаксисе оператора **CREATE PROCEDURE** допустимо использование блоков, которые описывают характеристики процедуры. Блоки



• **COMMENT**. Характеристика **COMMENT** — это указание MySQL. Она используется для описания процедуры, и задаётся информация отображается в выводе оператора SHOW CREATE PROCEDURE.

• **LANGUAGE**. Характеристика LANGUAGE указывает на то, что тело процедуры написано на SQL.

• **NOT DETERMINISTIC**. Это информационная характеристика. Процедура считается «детерминированной», если она всегда дает тот же результат для одних и тех же входных параметров, иначе она является недетерминированной».

• **CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA**

CONTAINS SQL означает, что в хранимой процедуре MySQL нет никаких заявлений, которые считывают или записывают данные. Например, заявления **SET @x = 1** или **DO RELEASE_LOCK('abc')**, они выполняются, но не считывают и не записывают данные. Это значение по умолчанию, если не указано другое значение характеристики.

• **NO SQL** означает, что процедура не содержит операторов SQL.

• **READS SQL DATA** — процедура содержит операторы, которые считывают данные (*например, SELECT*), но не содержит операторов, которые записывают данные.

• **MODIFIES SQL DATA-означает**, что подпрограмма содержит операторы, которые могут записывать данные (*например, INSERT или DELETE*).

• **SQL SECURITY {DEFINER | INVOKER}**

Значение **SQL SECURITY** может быть определено либо как **SQL SECURITY DEFINER**, либо как **SQL SECURITY INVOKER**. Оно указывает, выполняется ли подпрограмма с использованием привилегий аккаунта, указанного в условии **DEFINER**, или аккаунта пользователя, который осуществляют **MySQL** вызов хранимой процедуры. Этот аккаунт должен иметь разрешение на доступ к базе данных, с которой связана подпрограмма. Значение по умолчанию **DEFINER**. Пользователь, который запускает процедуру, должен иметь привилегию **EXECUTE**, если процедура выполняется в контексте безопасности **DEFINER**.

Все перечисленные блоки характеристик имеют значения по умолчанию.
Следующие два оператора дают одинаковый результат:

```
• mysql> CREATE PROCEDURE job_data()
-> SELECT * FROM JOBS; $$
Query OK, 0 rows affected (0.00 sec)
то же самое, что:
• mysql> CREATE PROCEDURE new_job_data()
• -> COMMENT ''
• -> LANGUAGE SQL
• -> NOT DETERMINISTIC
• -> CONTAINS SQL
• -> SQL SECURITY DEFINER
• -> SELECT * FROM JOBS;
• -> $$
• Query OK, 0 rows affected (0.26 sec)
```

10.5. MySQL: составные операторы

Составной оператор представляет собой блок, который может содержать другие блоки: объявления переменных, обработчиков состояний и курсоров, конструкции управления потоками данных, циклы и условные тесты. В версии **MySQL 5.6** существуют следующие составные операторы:

• Составной оператор **BEGIN … END**,
• Метки операторов;
• **DECLARE**,
• Переменные в хранимых программах;
• Операторы контроля потока данных;
• Курсоры;
• Обработчики условий.

Рассмотрим первые четыре оператора, связанные с параметрами оператора **CREATE PROCEDURE**.

5.1. Синтаксис составного оператора BEGIN … END

Он используется, когда нужно разместить в пределах подпрограммы (*например, хранимой процедуры MySQL, функции, триггера или события*) более одного оператора. Синтаксис следующий:

• [метка_начала:] BEGIN
• [список_операторов]

• END [метка_конца])

список_операторов: один или несколько операторов, завершающихся точкой с запятой (;). Сам по себе список операторов не является обязательным, поэтому пустой оператор **BEGIN END** является действительным.

5.2. Метки операторов

Метки — это разрешения на выполнение для блоков **BEGIN** ... **END** и операторов цикла **REPEAT** и **WHILE**. Синтаксис следующий:

• [метка_начала:] BEGIN
• [список_операторов]
• END [метка_конца]
• [метка_начала:] LOOP
• список_операторов
• END LOOP [метка_конца]
• [метка_начала:] REPEAT
• список_операторов
• UNTIL search_condition
• END REPEAT [метка_конца]
• [метка_начала:] WHILE условие_поиска
• DO список_операторов
• END WHILE [метка_конца]

5.3. Оператор DECLARE

Используется для определения различных локальных элементов в **MySQL** при создании хранимой процедуры. Например, локальных переменных, условий, обработчиков, курсоров. **DECLARE** используется только внутри составного оператора **BEGIN** ... **END** и должен находиться в его начале перед всеми остальными операторами.

Для объявлений существуют следующие правила:

• **Объявления курсоров должны размещаться перед объявлениями обработчиков;**

• **Объявления переменных и условий должны размещаться перед объявлениями курсоров или обработчиков**.

Переменные в хранимых программах. Хранимые программы пользуют оператор **DECLARE** для определения локальных переменных. оцедуры и функции могут при объявлении принимать параметры, которые ениваются значениями между подпрограммой и вызывающим ее агентом.

Объявление переменной:

• DECLARE имя_переменной [, имя_переменной] ... type [DEFAULT ение]

Чтобы предоставить значение для переменной по умолчанию, используется блок **DEFAULT**. Значение может быть задано как выражение, это не обязательно должна быть константа. Если блок **DEFAULT** отсутствует, начальное значение равно **NULL**.

Пример - Локальные переменные. Локальные переменные объявляются внутри хранимых процедур **MySQL**. Они действительны только в пределах блока **END** ... **BEGIN**, в котором они объявлены. Локальные переменные могут содержать любой тип данных **SQL**. В следующем примере показано использование локальных переменных в хранимой процедуре:

```
DELIMITER $$
CREATE PROCEDURE my_procedure_Local_Variables()
BEGIN  /* объявление локальной переменной */
DECLARE  a  INT  DEFAULT  10;
использование локальной переменной */
  SET a = a + 100;  SET b = 2;  SET c = a + b;  BEGIN    /* локальная
переменная во вложенном блоке */
  DECLARE c INT;          SET c = 5;     /* локальная переменная имеет
приоритет по сравнению с переменной с       тем же именем, объявленной
в закрытом блоке. */
  SELECT a, b, c;  END;  SELECT a, b, c;
END$$
```

Выполните процедуру:

```
• mysql> CALL my_procedure_Local_Variables();
• +------+------+------+
• | a    | b    | c    |
• +------+------+------+
• | 110 |  2 |  5 |
• +------+------+------+
• 1 row in set (0.00 sec)
• +------+------+------+
• | a    | b    | c    |
• +------+------+------+
• | 110 |  2 | 112 |
• +------+------+------+
• 1 row in set (0.01 sec)
• Query OK, 0 rows affected (0.03 sec)
```

Пример: пользовательские переменные. В хранимых процедурах **MySQL** обращение к пользовательским переменным происходит через символ **амперсанда (@)** перед именем пользовательской переменной (*например*, **@x** *и* **@y**). В следующем примере показано использование пользовательских переменных внутри хранимой процедуры:

```
• DELIMITER $$
• CREATE PROCEDURE my_procedure_User_Variables()
```

```
• BEGIN SET @x = 15; SET @y = 10;
• SELECT @x, @y, @x-@y;
• END$$
Выполнения процедуру:
mysql> CALL my_procedure_User_Variables();
+------+------+-------+
| @x   | @y   | @x-@y |
+------+------+-------+
|  15  |  10  |   5   |
+------+------+-------+
1 row in set (0.04 sec)
Query OK, 0 rows affected (0.05 sec)
```

Ниже приводится синтаксис **CREATE PROCEDURE** для параметров:

```
• CREATE [DEFINER = {пользователь | CURRENT_USER }]
PROCEDURE имя_процедуры ([параметр_процедуры[,...]])
[характеристики ...] тело_подпрограммы
```

• параметр_процедуры: [**IN | OUT | INOUT**] имя_параметра type

Варианты синтаксиса:

• **CREATE PROCEDURE** имя_процедуры () ...

• **CREATE PROCEDURE** имя_процедуры ([**IN**] **имя_параметра type**)...

• **CREATE PROCEDURE** имя_процедуры ([**OUT**] **имя_параметра type**)...

• **CREATE PROCEDURE** имя_процедуры ([**INOUT**] **имя_параметра type**)...

В первом примере список параметров пуст.

Во втором примере параметр **IN** передает значение в процедуру. Эта процедура может изменить значение. Но, когда процедура возвращает значение, оно не будет видно для вызывающего агента.

В третьем примере параметр **OUT** передает значение из процедуры обратно вызывающему агенту. Его начальное значение в процедуре **NULL**, и, когда процедура возвращает значение, оно видно вызывающему агенту.

В четвертом примере параметр **INOUT** инициализируется вызывающим агентом, он может быть изменен процедурой, и когда процедура возвращает значение, любые изменения, произведенные **MySQL** хранимой процедурой, будут видны вызывающему агенту.

В процедуре каждый параметр по умолчанию является параметром **IN**. Чтобы изменить, это используйте перед именем параметра ключевое слово **OUT** или **INOUT**.

Пример параметра IN. В следующей процедуре использован параметр «*var1*» (*тип целое число*), который принимает число от пользователя. В теле процедуры есть оператор **SELECT**, который выбирает строки из таблицы «*jobs*». Количество строк указывается пользователем. Ниже приводится процедура:


```
• mysql> CREATE PROCEDURE my_proc_IN (IN var1 INT)
• -> BEGIN
• -> SELECT * FROM jobs LIMIT var1;
• -> END$$
• Query OK, 0 rows affected (0.00 sec)
```

Чтобы выбрать первые две строки из таблицы «*jobs*» выполняется следующая команда:

```
• mysql> CALL my_proc_in(2)$$
```

• первые пять строк из таблицы «*jobs*«:

```
• mysql> CALL my_proc_in(5)$$
```

Пример параметра OUT. Дальше представлен **MySQL** хранимой процедуры пример, в котором используется параметр **OUT**. В рамках процедуры **MySQL функция MAX()** извлекает максимальную зарплату из столбца **MAX_SALARY** таблицы «*jobs*«:

```
• mysql> CREATE PROCEDURE my_proc_OUT (OUT highest_salary INT)
• -> BEGIN
• -> SELECT MAX(MAX_SALARY) INTO highest_salary FROM JOBS;
• -> END$$
• Query OK, 0 rows affected (0.00 sec)
```

В теле процедуры параметр получает самую высокую зарплату из столбца **MAX_SALARY**. После вызова процедуры слово **OUT** сообщает **СУБД**, что значение исходит от процедуры. **highest_salary** — это имя выходного параметра и в операторе **CALL** мы передали его значение переменной сеанса с именем **@M**:

```
• mysql> CALL my_proc_OUT(@M)$$
Query OK, 1 row affected (0.03 sec)
mysql> SELECT @M$$
+-------+
| @M    |
+-------+
| 40000 |
+-------+
1 row in set (0.00 sec)
```

Пример параметра INOUT. В следующем примере показана простая хранимая процедура **MySQL**, которая использует параметр **INOUT** и параметр **IN**. Пользователь предоставляет ‘**M**‘ или ‘**F**‘ через параметр **IN** (***emp_gender***) для подсчета количества сотрудников мужского или женского пола из таблицы **user_details**. Параметр **INOUT** (***mfgender***) возвращает результат пользователю. Вот код и результат выполнения процедуры:

```
• mysql> CREATE PROCEDURE my_proc_INOUT (INOUT mfgender INT,
IN emp_gender CHAR(1))
• -> BEGIN
```

```
-> SELECT COUNT(gender) INTO mfgender FROM user_details WHERE
gender = emp_gender;
-> END$$
Query OK, 0 rows affected (0.00 sec)
```

Проверяем количество сотрудников мужского и женского пола, указанной таблице:

```
mysql> CALL my_proc_INOUT(@C,'M')$$
Query OK, 1 row affected (0.02 sec)
mysql> SELECT @C$$
+------+
| @C   |
+------+
|  3   |
+------+
1 row in set (0.00 sec)
mysql> CALL my_proc_INOUT(@C,'F')$$
Query OK, 1 row affected (0.00 sec)
mysql> SELECT @C$$
+------+
| @C   |
+------+
|  1   |
+------+
1 row in set (0.00 sec)
```

Контрольные вопросы

1. Хранимая процедура SQL. Два вида подпрограмм SQL.
2. В чем преимущество хранимых процедур?
3. Создание процедуры CREATE PROCEDURE. Синтаксис.
4. Параметры хранимых процедур SQL.
5. Блоки характеристик characteristic хранимых процедур SQL.
6. Составные операторы хранимых процедур SQL.
7. Переменные в хранимых программах SQL. Оператор DECLARE.
8. Переменные в хранимых программах. Параметры IN, OUT, INOUT.

11.1. *Понятие транзакции*

Транзакция – это последовательность операций, проводимых над БД, рассматриваемых как единое целое и переводящих БД из одного корректного состояния в другое непротиворечивое состояние. Количество операций, входящих в транзакцию, может быть любым от одной до сотен, тысяч. Разработчик решает, какие команды должны выполняться как одна транзакция, а какие могут быть разбиты на несколько последовательно выполняемых транзакций.

При выполнении транзакции СУБД должна обеспечить обработку некоторых команд, входящих в транзакцию, так, чтобы гарантировать правильность и надежность работы системы.

Транзакция должна удовлетворять ACID – требованиям.

11.2. *ACID – требования*

ACID – требования гарантируют правильность и надежность работы системы и является фундаментальные свойства систем обработки транзакций

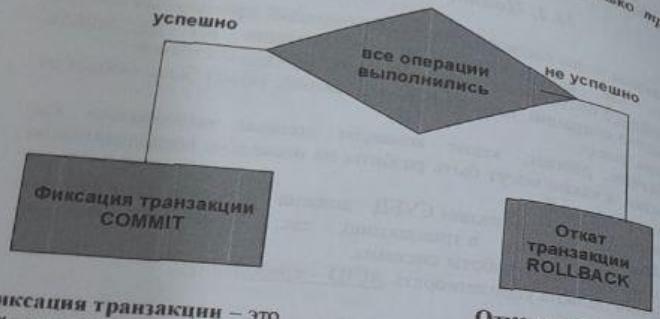
- **Atomic** (атомарность) - Транзакция не может выполниться частично, либо все, либо ничего
- **Consistency (согласованность** - После выполнения транзакции все данные должны находиться в согласованном состоянии .
- **Isolation (изолированность)** - Транзакция должна быт автономной и воздействовать на другие транзакции или зависыть от них.
- **Durability (устойчивость)** - После завершения транзакции, внесенные изменения останутся неизменными.

Варианты завершения транзакций. **2 варианта завершения транзакций:**

Блокировки. Повышение эффективности работы при использовании небольших *транзакций* связано с тем, что при выполнен и *транзакции сервер* накладывает на данные ***блокировки***.

- ***Блокировкой*** называется временное ограничение на выполнение некоторых операций обработки данных. *Блокировка* может быть наложена как на отдельную строку таблицы, так и на всю базу данных.
- ***Управлением блокировками*** на сервере занимается менеджер блокировок, контролирующий их применение и разрешение конфликтов. *Транзакции* и *блокировки* тесно связаны друг с другом. *Транзакции* накладывают блокировки на данные, чтобы обеспечить выполнение

требований *ACID*. Без использования блокировок несколько транзакций могли бы изменять одни и те же данные.



Фиксация транзакции – это **действия**, обеспечивающие **сохранение** на диске изменений БД, сделанные в процессе выполнения транзакции

Откат транзакции – это **действия**, обеспечивающие аннулирование **всех** изменений БД, сделанные в процессе выполнения транзакции

Блокировка представляет собой метод *управления параллельными процессами*, при котором *объект БД* не может быть модифицирован без ведома *транзакции*, т.е. происходит блокирование доступа к объекту со стороны других *транзакций*, чем исключается непредсказуемое изменение объекта. Различают два вида *блокировки*:

- *блокировка* записи – *транзакция* блокирует строки в таблицах таким образом, что запрос другой *транзакции* к этим строкам будет *отменен*;
- *блокировка* чтения – *транзакция* блокирует строки так, что запрос со стороны другой *транзакции* на *блокировку* записи этих строк будет отвергнут, а на *блокировку* чтения – принят.

В *СУБД* используют протокол доступа к данным, позволяющий избежать проблемы параллелизма. Его суть заключается в следующем:

- *транзакция*, результатом действия которой на строку данных в таблице является ее извлечение, обязана наложить *блокировку* чтения на эту строку;
- *транзакция*, предназначенная для модификации строки данных, закладывает на нее *блокировку* записи;
- если запрашиваемая *блокировка* на строку отвергается из-за уже имеющейся *блокировки*, то *транзакция* переводится в режим ожидания до тех пор, пока *блокировка* не будет снята;

- *блокировка* записи сохраняется вплоть до конца выполнения транзакции.
- Решение ***проблемы параллельной обработки*** БД заключается в том, что строки таблиц блокируются, а последующие *транзакции*, модифицирующие эти строки, отвергаются и переводятся в режим ожидания. В связи со свойством сохранения целостности БД *транзакции*, подходящими единицами изолированности являются *транзакции*. Если каждый сеанс взаимодействия с базой данных реализуется *транзакцией*, то пользователь работал с ней в одиночку.

11.3. Проблемы при выполнении транзакций

Если в системе управления базами данных не реализованы *механизмы* блокирования, то при одновременном чтении и изменении одних и тех же данных несколькими пользователями могут возникнуть следующие проблемы одновременного доступа:

- проблема последнего изменения (*lost update*) возникает, когда несколько пользователей изменяют одну и ту же строку, основываясь на ее начальном значении, тогда часть данных будет потеряна, т.к. каждая последующая *транзакция* перезапишет изменения, сделанные предыдущей. Выход из этой ситуации заключается в последовательном внесении изменений;
- проблема ***"грязного" чтения*** (*dirty read*) возможна в том случае, если пользователь выполняет сложные операции обработки данных, требующие множественного изменения данных перед тем, как они обретут логически верное состояние. Если во время изменения данных другой пользователь будет считывать их, то может оказаться, что он получит логически неверную информацию. Для исключения подобных проблем необходимо производить считывание данных после окончания всех изменений;
- проблема ***неповторяемого чтения*** (*non-repeatable read*) является следствием неоднократного считывания *транзакцией* одних и тех же данных. Во время выполнения первой *транзакции* другая может внести в данные изменения, поэтому при повторном чтении первая *транзакция* получит уже иной набор данных, что приводит к нарушению их целостности или логической несогласованности;
- проблема чтения ***фантомов*** (*phantom reads*) появляется после того, как одна *транзакция* выбирает данные из таблицы, а другая вставляет или удаляет строки до завершения первой. Выбранные из таблицы значения будут некорректны.

Для решения перечисленных проблем в специально разработанном стандарте определены четыре *уровня блокирования*. Уровень *изоляции*

транзакции определяет, могут ли другие (конкурирующие) *транзакции* вносить изменения в данные, измененные текущей *транзакцией*, а также может ли текущая *транзакция* видеть изменения, произведенные конкурирующими *транзакциями*, и наоборот. Каждый последующий уровень поддерживает требования предыдущего и налагает дополнительные ограничения: **Serializable, Repeatable Read, Read Committed, Read Uncommitted**.

Каждый последующий уровень поддерживает требования предыдущего и налагает дополнительные ограничения:

- *уровень 0* – запрещение "загрязнения" данных. Этот *уровень* требует, чтобы изменять данные могла только одна *транзакция*; если другой *транзакции* необходимо изменить те же данные, она должна ожидать завершения первой *транзакции*;
- *уровень 1* – запрещение *"грязного" чтения*. Если *транзакция* начала изменение данных, то никакая другая *транзакция* не сможет прочитать их до завершения первой;
- *уровень 2* – запрещение *неповторяемого чтения*. Если *транзакция* считывает данные, то никакая другая *транзакция* не сможет их изменить. Таким образом, при повторном чтении они будут находиться в первоначальном состоянии;
- *уровень 3* – запрещение *фантомов*. Если *транзакция* обращается к данным, то никакая другая *транзакция* не сможет добавить новые или удалить имеющие строки, которые могут быть считаны при выполнении *транзакции*. Реализация этого *уровня блокирования* выполняется путем использования блокировок диапазона ключей. Подобная *блокировка* накладывается не на конкретные строки таблицы, а на строки, удовлетворяющие определенному логическому условию.

Уровни изоляции транзакций

- **Serializable** – нельзя обращаться к данным, обрабатываемым другой транзакцией.
- **Repeatable Read** – нельзя обращаться к обновленным или удаленным данным, но можно к добавленным.
- **Read Committed** – можно обращаться к зафиксированным данным.
- **Read Uncommitted** – можно обращаться к любым обновленным и не зафиксированным данным.
- Snapshot – каждая транзакция работает со своей версией данных.

SET TRANSACTION ISOLATION LEVEL
{ READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SNAPSHOT | SERIALIZABLE }

Уровень изоляции	Lost update- потерянное обновление	Dirty read- «Грязное» чтение	Non-repeatable read- Неповторяющееся чтение	Phantom read- Фантомное чтение
Serializable				
Repeatable Read				
Read Committed				
Read Uncommitted				
Snapshot				

Рис. Уровни изоляции транзакций и ошибки целостности.

READ UNCOMMITTED

- Не устанавливаются блокировки на чтение
- Транзакции могут считывать измененные другими транзакциями, но не зафиксированные строки. Это позволяет считывать незафиксированные изменения, которые называются чтением«грязных» данных.
- Значения в данных могут быть изменены и до окончания транзакции строки могут появляться и исчезать в наборе данных. Это наименьшее ограничение уровней изоляции.
- Нельзя считывать данные, которые были изменены другими транзакциями, но еще не были зафиксированы - «грязные» данных.
- Считанные данные могут быть изменены другими транзакциями во время работы текущей транзакции, результатом чего будет неповторяемое чтение или недействительные данные.
- Этот режим в SQL Server установлен по умолчанию.

REPEATABLE READ

- Нельзя считывать данные, которые были изменены, но еще не зафиксированы другими транзакциями.
- Совмещаемые блокировки применяются ко всем считываемым данным и сохраняются до завершения. Это запрещает другим транзакциям изменять строки, считанные текущей транзакцией.

•Другие транзакции могут вставлять новые строки, соответствующие условиям поиска текущей транзакции. При повторном чтении диапазона могут появиться новые строки «фантомы».

SNAPSHOT

•Копирует БД на момент начала транзакции. Текущая транзакция не видит изменений данных, произведенных другими транзакциями после запуска текущей транзакции.

•Считывание данных транзакциями моментальных снимков не блокирует запись данных другими транзакциями.

•Если транзакция пытается записать данные, измененные кем-то после ее начала, то она обрывается.

•Перед запуском транзакции, использующей уровень изоляции моментальных снимков, необходимо установить параметр базы данных ALLOW_SNAPSHOT_ISOLATION в ON.

•Транзакция, работающая с уровнем изоляции моментального снимка, может просматривать внесенные ею изменения.

SERIALIZABLE

•Нельзя считывать данные, которые были изменены другими транзакциями, но еще не были зафиксированы.

•Другие транзакции не могут изменять данные, считываемые текущей транзакцией.

•Другие транзакции не могут вставлять новые строки со значениями ключа, считываемых текущей транзакции, до ее завершения. При повторном считывании будет тот же самый набор строк.

•Блокировки считанного диапазона сохраняются до завершения транзакции.

•Из-за низкого параллелизма этот параметр рекомендуется использовать только при необходимости.

11.4. Управлением транзакциями

Под ***управлением транзакциями*** понимается способность управлять различными операциями над данными, которые выполняются внутри реляционной СУБД. Прежде всего, имеется в виду выполнение операторов INSERT, UPDATE и DELETE. Например, после создания таблицы (выполнения оператора CREATE TABLE) не нужно фиксировать результат: создание таблицы фиксируется в базе данных автоматически. Точно так же с

помощью операции транзакции не удастся восстановить только что удаленную оператором DROP TABLE таблицу.

После успешного выполнения команд, заключенных в тело одной транзакции, немедленного выполнения команд, заключенных в тело немедленного завершения транзакции изменения данных не происходит. Для команды управления транзакциями существуют так называемые сохранить в базе данных все изменения, произошедшие в ходе ее выполнения, либо полностью их отменить.

Существуют три команды, которые используются для управления транзакциями:

•**COMMIT** – для сохранения изменений ;

•**ROLLBACK** – для отмены изменений ;

•**SAVEPOINT** – для установки особых точек возврата.

Виды определения транзакций

•Явное - Требуется явно указать команды начала и конца транзакций

•Автоматическое - Каждая команда рассматривается как отдельная транзакция,

•Неявное (подразумеваемое) - Транзакция начинается с первого оператора SQL и заканчивается явным указанием конца транзакции;

Транзакционные и блокирующие запросы.

MySQL поддерживает транзакции (в пределах данного сеанса клиента) через запросы:

•SET autocommit

•START TRANSACTION

•COMMIT

•ROLLBACK

Операторы управления транзакциями

SET autocommit. MySQL по умолчанию работает в режиме AUTOCOMMIT. Это означает, что если вы не начали транзакцию явным образом, каждый запрос автоматически выполняется в отдельной транзакции. Вы можете включить или отключить режим AUTOCOMMIT для текущего соединения, установив следующее значение конфигурационной переменной:

```
mysql> SHOW VARIABLES LIKE 'AUTOCOMMIT';
START TRANSACTION [transaction_characteristic [,
transaction_characteristic] ...]
transaction_characteristic: {
    WITH CONSISTENT SNAPSHOT
  | READ WRITE
```

| READ ONLY
}
BEGIN [WORK]
COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
SET autocommit = {0 | 1}
START TRANSACTION или BEGIN – применяется для того, чтобы:
Зафиксировать, что транзакция началась. Указать (при необходимости), какие объекты захватываются и уровень их блокировки
SAVEPOINT <NAME>
Указывает точку возврата, к которой можно откатиться при частичном откате транзакции.
SAVEPOINT имя_точки_сохранения
RELEASE SAVEPOINT <NAME>
Удаление успешно пройденной точки возврата
COMMIT – применяется для того, чтобы:
- сделать «постоянными» все изменения, сделанные в текущей транзакции (реально данные могут быть изменены несколько позже)
- очистить все точки сохранения данной транзакции
- завершить транзакцию
- освободить все блокировки данной транзакции

ROLLBACK – применяется для того, чтобы:
- отменить все изменения, внесённые начиная с момента начала транзакции или с какой-то точки сохранения (SAVEPOINT).
- очистить все точки сохранения данной транзакции
- завершить транзакцию
- освободить все блокировки данной транзакции

примеры:

```
BEGIN;
 INSERT INTO table1 VALUES (1);
 SAVEPOINT my_savepoint;
 INSERT INTO table1 VALUES (2);
 ROLLBACK TO SAVEPOINT my_savepoint;
 INSERT INTO table1 VALUES (3);
COMMIT;

BEGIN;
 INSERT INTO table1 VALUES (3);
 SAVEPOINT my_savepoint;
 INSERT INTO table1 VALUES (4);
 RELEASE SAVEPOINT my_savepoint;
COMMIT;
```



```
START TRANSACTION;
SELECT @A:=SUM(salary) FROM table1 WHERE type=1;
UPDATE table2 SET summary=@A WHERE type=1;
COMMIT;
```

Взаимоблокировки

Взаимоблокировка возникает тогда, когда две и более транзакции взаимно удерживают и запрашивают блокировку одних и тех же ресурсов, создавая циклическую зависимость. Такие состояния наблюдаются и в том случае, если транзакции пытаются заблокировать ресурсы в разном порядке. Они могут возникнуть, когда несколько транзакций блокируют одни и те же ресурсы. Для примера рассмотрим две транзакции, обращающиеся к таблице StockPrice.

• Транзакция № 1

```
START TRANSACTION;
UPDATE StockPrice SET close = 45.50 WHERE stock_id = 4 and date = '2002-05-01';
UPDATE StockPrice SET close = 19.80 WHERE stock_id = 3 and date = '2002-05-02';
COMMIT;
```

• Транзакция № 2

```
START TRANSACTION;
UPDATE StockPrice SET high = 20.12 WHERE stock_id = 3 and date = '2002-05-02';
UPDATE StockPrice SET high = 47.20 WHERE stock_id = 4 and date = '2002-05-01';
COMMIT;
```

Если вам не повезет, то каждая транзакция выполнит свой первый запрос и обновит строку данных, заблокировав ее. Затем все транзакции попытаются обновить вторую строку, но обнаружат, что та уже заблокирована. В итоге каждая транзакция будет до бесконечности ожидать окончания другой, пока не произойдет вмешательство извне, которое снимет взаимоблокировку.

Контрольные вопросы

1. Что такое транзакция. Понятия транзакции.
2. ACID – требования транзакции.
3. Блокировки. Назначения блокировки. Виды блокировок.
4. Проблемы при выполнении транзакций.
5. Решение проблемы при выполнении транзакций. Уровни изоляции транзакций.

6. Операторы SQL для управления транзакциями.
7. SQL операторы управления транзакциями: START TRANSACTION и SAVEPOINT.
8. SQL операторы управления транзакциями: COMMIT и ROLLBACK.
9. Что такое взаимоблокировка транзакций.

С развитием систем баз данных процедуры инсталляции и использования MySQL становятся все проще. Судя по всему, именно простота работы с MySQL стала основной причиной широкой ее популярности среди пользователей. Особенно это относится к тем из них, которые не являются, да и не желают быть программистами. Безусловно, знания компьютерного профессионала могут оказаться весьма полезными, но для успешного использования MySQL быть опытным программистом вовсе не обязательно.

12.1. Проблемы безопасности и система привилегий доступа MySQL

Система привилегий

Система привилегий MySQL обеспечивает пользователям возможность выполнять только те действия, которые им разрешены.

MySQL идентифицирует пользователя по имени хоста и по имени пользователя.

12.2. Синтаксис команд GRANT и REVOKE

Команды **GRANT** и **REVOKE** позволяют системным администраторам создавать пользователей MySQL, а также предоставлять права пользователям или лишать их прав на четырех уровнях привилегий:

Глобальный уровень

Глобальные привилегии применяются ко всем базам данных на указанном сервере. Эти привилегии хранятся в таблице mysql.user.

Уровень базы данных

Привилегии базы данных применяются ко всем таблицам указанной базы данных. Эти привилегии хранятся в таблицах mysql.db и mysql.host.

Уровень таблицы

Привилегии таблицы применяются ко всем столбцам указанной таблицы. Эти привилегии хранятся в таблице mysql.tables_priv.

Уровень столбца

Привилегии столбца применяются к отдельным столбцам указанной таблицы. Эти привилегии хранятся в таблице mysql.columns_priv.

- Синтаксис команд GRANT

```
GRANT
priv_type [(column_list)]
[,priv_type [(column_list)]] ...
```

```
    ON [object_type] priv_level
    TO user_or_role [, user_or_role] ...
    [AS user
     [WITH ROLE
        DEFAULT
      | NONE
      | ALL
      | ALL EXCEPT role [, role ] ...
      | role [, role ] ...
     ]
    ]

  GRANT PROXY ON user_or_role
    TO user_or_role [, user_or_role] ...
    [WITH GRANT OPTION]

  GRANT role [, role] ...
    TO user_or_role [, user_or_role] ...
    [WITH ADMIN OPTION]

  object_type: {
      TABLE
    | FUNCTION
    | PROCEDURE
  }

  priv_level: {
      *
    | *.*
    | db_name.*
    | db_name.tbl_name
    | tbl_name
    | db_name.routine_name
  }

  user_or_role: { user | role }
GRANT priv_type [(column_list)] [,priv_type [(column_list)]...]
  ON {tbl_name | * | *.* | db_name.*}
  TO user_name [IDENTIFIED BY 'password']
    [, user_name ...]
  [WITH GRANT OPTION]
REVOKE priv_type [(column_list)] [,priv_type [(column_list)]...]
  ON {tbl_name | * | *.* | db_name.*}
  FROM user_name [, user_name ...]
```


Если привилегии представляются пользователю, которого не существует, то этот пользователь создается.

Список возможных значений параметра priv_type:

ALL [PRIVILEGES]	Задает все простые привилегии, кроме WITH GRANT OPTION
ALTER	Разрешает использование ALTER TABLE
CREATE	Разрешает использование CREATE TABLE
CREATE TEMPORARY TABLES	Разрешает использование CREATE TEMPORARY TABLE
DELETE	Разрешает использование DELETE
DROP	Разрешает использование DROP TABLE
EXECUTE	Разрешает пользователю запускать хранимые процедуры (для MySQL 5.0)
FILE	Разрешает использование SELECT ... INTO OUTFILE и LOAD DATA INFILE
INDEX	Разрешает использование CREATE INDEX and DROP INDEX
INSERT	Разрешает использование INSERT
LOCK TABLES	Разрешает использование LOCK TABLES на таблицах, для которых есть привилегия SELECT
PROCESS	Разрешает использование SHOW FULL PROCESSLIST
REFERENCES	Зарезервировано для использования в будущем
RELOAD	Разрешает использование FLUSH
REPLICATION CLIENT	Предоставляет пользователю право запрашивать местонахождение головного и подчиненных серверов
REPLICATION SLAVE	Необходимо для подчиненных серверов при репликации (для чтения информации из бинарных журналов головного сервера)
SELECT	Разрешает использование SELECT
SHOW DATABASES	SHOW DATABASES выводит все базы данных
SHUTDOWN	Разрешает использование mysqladmin shutdown
SUPER	Позволяет установить одно соединение (один раз), даже если достигнуто значение max_connections, и запускать команды CHANGE MASTER, KILL thread, mysqladmin debug, PURGE MASTER LOGS и SET GLOBAL
UPDATE	Разрешает использование UPDATE
USAGE	Синоним для "без привилегий"
GRANT OPTION	Синоним для WITH GRANT OPTION

Для таблицы можно указать только следующие значения: SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, GRANT OPTION, INDEX и ALTER.

Для столбца можно указать только следующие значения: SELECT, INSERT и UPDATE.

Глобальные привилегии можно задать, используя синтаксис ON *.*, а привилегии базы данных - при помощи синтаксиса ON db_name.*.

Если указать ON * при открытой текущей базы данных, то привилегии будут заданы для этой базы данных. (**Предупреждение:** если указать ON * при отсутствии открытой текущей базы данных, это повлияет на глобальные привилегии)

Примечание. В операторе GRANT шаблонные символы % не допускаются в определении имени баз данных.

Имя пользователя (user_name) можно указать в форме user@host. Если необходимо указать строку user, в которой содержатся специальные символы (такие как '-') или строку host, в которой содержатся специальные или групповые символы (такие как '%'), необходимо заключить имя указанного компьютера или пользователя в кавычки (например, 'test-user'@'test-hostname').

В имени удаленного компьютера также можно указывать групповые символы. Например, user@'%.loc.gre' относится к user всех удаленных компьютеров домена loc.gre, а user@'144.155.166.%' относится к user всех удаленных компьютеров подсети 144.155.166.

Простая форма user является синонимом для user@'%'.

Привилегии для таблицы или столбца формируются при помощи логического оператора OR из привилегий каждого из четырех уровней. Например, если в таблице mysql.user указано, что у пользователя есть глобальная привилегия SELECT, эта привилегия не отменяется на уровне базы данных, таблицы или столбца.

Привилегии для столбца могут быть вычислены следующим образом:

глобальные привилегии
OR привилегии базы данных
OR привилегии таблицы
OR привилегии столбца

В большинстве случаев права пользователя определяются только на одном уровне привилегий, поэтому обычно эта процедура не настолько сложна, как описано выше.

Если привилегии предоставляются сочетанию пользователь@удаленный компьютер, которое отсутствует в таблице mysql.user, то в последнюю добавляется запись, которая остается в таблице до тех пор, пока не будет удалена при помощи команды DELETE.

Т.е. команда GRANT может создавать записи user в таблице, но команда REVOKE не может их удалить. Это необходимо делать при помощи команды DELETE.

12.3. GRANT примеры

Синтаксис короткий:

• **mysql> GRANT <тип привилегий> ON <объект> TO <пользователь> [IDENTIFIED BY <пароль>] <дополнительные опции>;**

Например, эта команда предоставляет права доступа пользователю и, если его не существует, создает его:

• mysql> GRANT ALL PRIVILEGES ON *.* TO 'dbuser'@'localhost' IDENTIFIED BY 'password' WITH GRANT OPTION;

Описание команды:

• **ALL PRIVILEGES:** предоставляет полные права на использование данных. *.* : права предоставляются на все базы и все таблицы.

• **dbuser:** имя учетной записи.

• **localhost:** доступ для учетной записи будет предоставлен только с локального компьютера.

• **password:** пароль, который будет задан пользователю.

• **WITH GRANT OPTION:** дает пользователю способность дать другим пользователям любые привилегии, которые пользователь имеет на указанном уровне привилегии..

Рассмотрим примеры предоставления привилегий на таблицу в MySQL. Например, что бы предоставить привилегии SELECT, INSERT, UPDATE и DELETE в таблице contacts, с именем пользователя dbadam, вы должны запустить следующий оператор GRANT:

• GRANT SELECT, INSERT, UPDATE, DELETE ON contacts TO 'dbadam'@'localhost';

Предоставить все разрешения, кроме GRANT OPTION, пользователю с именем 'dbadam'.

• GRANT ALL ON contacts TO 'dbadam'@'localhost';

Предоставить только доступ SELECT в таблице contacts всем пользователям

• GRANT SELECT ON contacts TO '*'@'localhost';

• GRANT SELECT, UPDATE (birthday) ON people TO root3@localhost;

12.4. Отменить привилегии на таблицу

• Синтаксис:

• REVOKE privileges ON object FROM user;

Отменить привилегии DELETE и UPDATE в таблице contacts пользователю с именем dbadam:

• REVOKE DELETE, UPDATE ON contacts
FROM 'foobar'@'localhost';
Отменить все разрешения (кроме GRANT OPTION) в таблице для пользователя с именем foobar.
• REVOKE ALL ON contacts FROM 'foobar'@'localhost';

12.5. Предоставить привилегии на функции / процедуры

Синтаксис предоставления привилегий EXECUTE для функции / процедуры в MySQL:

• **GRANT EXECUTE ON [PROCEDURE | FUNCTION] object TO user;**
➢ EXECUTE — возможность выполнения функции или процедуры.
➢ PROCEDURE — используется, когда привилегия предоставляется процедуре в MySQL.
➢ FUNCTION — используется, когда привилегия предоставляется функции в MySQL.
➢ object — имя объекта базы данных, для которого предоставляются привилегии. В случае предоставления привилегий EXECUTE для функции или процедуры это будет имя функции или имя процедуры.
➢ user — имя пользователя, которому будут предоставлены привилегии EXECUTE.RE [FUNCTION] object TO user;

Пароль пользователя будет назначаться оператором IDENTIFIED BY, если он указан. Если у пользователя уже есть пароль, то этот пароль будет заменен новым.

Если при создании нового пользователя не указать оператор IDENTIFIED BY, будет создан пользователь без пароля. Это ненадежно с точки зрения безопасности.

Пароли также можно задавать при помощи команды SET PASSWORD.

SET PASSWORD FOR root@localhost = PASSWORD('new_password');

Если у пользователя нет никаких привилегий для таблицы, то таблица не отображается, когда пользователь запрашивает список таблиц (например, при помощи оператора SHOW TABLES).

Оператор **WITH GRANT OPTION** предоставляет пользователю возможность наделять других пользователей любыми привилегиями, которые он сам имеет на указанном уровне привилегий. При предоставлении привилегии GRANT необходимо проявлять осмотрительность, так как два пользователя с разными привилегиями могут объединить свои привилегии!

Нельзя предоставить другому пользователю привилегию, которой нет у вас самого. Привилегия GRANT позволяет предоставлять только те привилегии, которыми вы обладаете.

Учтите, что если пользователю назначена привилегия GRANT на определенном уровне привилегий, то все привилегии, которыми этот пользователь уже обладает (или которые будут ему назначены в будущем!) на этом уровне, также могут назначаться этим пользователем.

Предположим, пользователю назначена привилегия INSERT в базе данных. Если потом в базе данных назначить привилегию SELECT и указать WITH GRANT OPTION, пользователь сможет назначать не только привилегию SELECT, но также и INSERT. Если затем в базе данных предоставить пользователю привилегию UPDATE, пользователь сможет после этого назначать INSERT, SELECT и UPDATE.

Не следует назначать привилегии ALTER обычным пользователям. Это дает пользователю возможность разрушить систему привилегий путем переименования таблиц!

Изменения в таблицах назначения привилегий, которые осуществляются при помощи команд GRANT и REVOKE, обрабатываются сервером немедленно. Если изменять таблицы назначения привилегий вручную (используя команды INSERT, UPDATE и т.д.), необходимо выполнить оператор **FLUSH PRIVILEGES** или **mysqladmin flush-privileges**, чтобы указать серверу на необходимость перезагрузки таблиц назначения привилегий.

12.6. Добавление новых пользователей в MySQL

Пользователей можно добавлять тремя различными способами:
1) при помощи команды CREATE USER;
2) при помощи команды GRANT;
3) напрямую в таблицы назначения привилегий.

Предпочтительнее использовать команду GRANT - этот способ проще и дает меньше ошибок.

Существует также большое количество программ (таких как phpmyadmin), которые служат для создания и администрирования пользователей.

Команда CREATE USER

CREATE USER user [IDENTIFIED BY [PASSWORD] 'password'] [, user [IDENTIFIED BY [PASSWORD] 'password']] .

The CREATE USER statement creates new MySQL accounts. To use it, you must have the global CREATE USER privilege or the INSERT privilege for the mysql database.

For each account, CREATE USER creates a new row in the mysql.user table that has no privileges. ***An error occurs if the account already exists***.

Each account is named using the same format as for the GRANT statement; for example, 'jeffrey'@'localhost'. The user and host parts of the account name correspond to the User and Host column values of the user table row for the account.

Оператор CREATE USER создает учетную запись базы данных, которая позволяет вам входить в базу данных MySQL. Синтаксис для оператора CREATE USER в MySQL:

```
CREATE USER user_name
IDENTIFIED BY [ PASSWORD ] 'password_value';
```

Параметры или аргументы

user_name — имя учетной записи базы данных, которую вы хотите создать.

PASSWORD — необязательный. Указываете ли вы этот параметр или нет, оператор CREATE USER будет вести себя одинаково.

password_value — пароль для назначения имени пользователя.

Команда CREATE USER примеры

```
CREATE USER
  'saman'@'localhost' IDENTIFIED BY '123456789';
```

В этом примере оператор CREATE USER создаст нового пользователя с именем saman в базе данных MySQL, пароль которого является '123456789'.

Создание нескольких пользователей

Вы можете создавать более одного пользователя за раз в MySQL. Вы можете использовать оператор CREATE USER для создания нескольких пользователей разделенных запятой комбинацией каждого user/password. Например:

```
CREATE USER
  'samvel'@'localhost' IDENTIFIED BY 'klondike',
  'serg'@'localhost' IDENTIFIED BY 'titidog';
```

Этот пример CREATE USER создаст двух пользователей в MySQL. Первый пользователь будет называться samvel с паролем 'klondike', а второй пользователь будет называться serg с паролем 'titidog'.

Использование Hash значения для пароля

В приведенных выше примерах отображается пароль открытого текста. У вас также есть возможность задать хэш-значение для пароля. Например:

```
CREATE USER 'samvel'@'localhost' IDENTIFIED BY
'*39C549BDECFBA8AFC3CE6B948C9359A0ECE08DE2';
```

Этот пример CREATE USER создаст нового пользователя samvel в базе данных MySQL с хэш-значением пароля.

```
SELECT PASSWORD('xyz');
Результат: *39C549BDECFBA8AFC3CE6B948C9359A0ECE08DE2
SELECT PASSWORD('password');
Результат: *2470C0C06DEE42FD1618BB99005ADCA2EC9D1E19

SELECT PASSWORD('google');
#Результат: *288CAC5A9F4E53A9DCEA23A3EDCE42C695CF48B9
SELECT PASSWORD(NULL);
#Результат: NULL
```



```
mysql> GRANT ALL PRIVILEGES ON *.* TO monty@localhost
    -> IDENTIFIED BY 'some_pass' WITH GRANT OPTION;
mysql> GRANT ALL PRIVILEGES ON *.* TO monty@"%"
    -> IDENTIFIED BY 'some_pass' WITH GRANT OPTION;
mysql> GRANT RELOAD,PROCESS ON *.*
    -> TO admin@localhost;
mysql> GRANT USAGE ON *.* TO dummy@localhost;
```

Эти команды GRANT создают **трех** новых пользователей.

monty

Полноценный суперпользователь - он может подсоединяться к серверу откуда угодно, но должен использовать для этого пароль some_pass. Обратите внимание на то, что мы должны применить операторы GRANT как для monty@localhost, так и для monty@"%". Если не добавить запись с localhost, запись анонимного пользователя для localhost, которая создается при помощи mysql_install_db, будет иметь преимущество при подсоединении с локального компьютера, так как в ней указано более определенное значение для поля Host, и она расположена раньше в таблице user.

admin

Пользователь, который может подсоединяться с localhost без пароля; ему назначены административные привилегии RELOAD и PROCESS. Эти привилегии позволяют пользователю запускать команды mysqladmin reload, mysqladmin refresh и mysqladmin flush-*, а также mysqladmin processlist. Ему не назначено никаких привилегий, относящихся к базам данных (их можно назначить позже, дополнительно применив оператор GRANT).

dummy

Пользователь, который может подсоединяться к серверу без пароля, но только с локального компьютера. Все глобальные привилегии установлены в значение 'N'-тип привилегии USAGE, который позволяет создавать пользователей без привилегий. Предполагается, что относящиеся к базам данных привилегии будут назначены позже.

Непосредственное создание пользователя при помощи команды INSERT

```
mysql> INSERT INTO user VALUES('localhost','monty',
    -> PASSWORD('some_pass'),
    ->      'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y');
mysql>
```

user

INTO

INSERT

```
VALUES('%','monty',PASSWORD('some_pass'),
    -> 'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO user SET Host='localhost',User='admin',
    -> Reload_priv='Y', Process_priv='Y';
```

```
mysql> GRANT
    -> SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
    -> ON bank.*
    -> TO custom@localhost
    -> IDENTIFIED BY 'student';
mysql> GRANT
    -> SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
    -> ON exp.*
    -> TO custom@whitehouse.ru
    -> IDENTIFIED BY 'student';
mysql> GRANT
    -> SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
    -> ON customer.*
    -> TO custom@'%'
    -> IDENTIFIED BY 'student';
```

Примечание. Команда SHOW PRIVILEGES показывает список системных привилегий, которые поддерживаются сервером MySQL.

```
mysql> show privileges;
+-----------+---------------------------+--------------------------------------------------------+
| Privilege | Context                   | Comment                                                |
+-----------+---------------------------+--------------------------------------------------------+
| Select    | Tables                    | To retrieve rows from                                  |
| Insert    | Tables                    | To insert data into                                    |
| Update    | Tables                    | To update existing                                     |
| Delete    | Tables                    | To delete existing                                     |
| Index     | Tables                    | To create or drop                                      |
| Alter     | Tables                    | To alter the                                           |
| Create    | Databases,Tables,Indexes  | To create new databases and                            |
| Drop      | Databases,Tables          | To drop databases and tables                           |
| Grant     | Databases,Tables          | To give to other users those privileges you possess    |
| References| Databases,Tables          | To have references on tables                           |
| Reload    | Server Admin              | To reload or refresh tables,                           |
logs and privileges
| Shutdown  | Server Admin              | To shutdown the                                        |
| Process   | Server Admin              | To view the plain text of currently executing queries  |
| File      | File access on server     | To read and write files on the server                  |
+-----------+---------------------------+--------------------------------------------------------+
14 rows in set (0.00 sec)
```

13.1. Соединители и API MySQL

MySQL Connectors обеспечивают подключение к серверу MySQL для клиентских программ. API обеспечивают низкоуровневый доступ к протоколу MySQL и ресурсам MySQL. Как Connectors, так и API-интерфейсы позволяют подключать и выполнять операторы MySQL из другого языка или среды, включая экземпляры ODBC, Java (JDBC), Perl, Python, PHP, Ruby и Руководство по соединителям и API

MySQL Connectors. Oracle разрабатывает несколько соединителей:

- Connector / C ++ позволяет приложениям C ++ подключаться к MySQL.
- Connector / J обеспечивает поддержку драйвера для подключения MySQL из приложений Java с использованием стандартного API подключения к базе данных Java (JDBC).
- Connector / NET позволяет разработчикам создавать приложения .NET, которые подключаются к MySQL. Connector / NET реализует полнофункциональный интерфейс ADO.NET и обеспечивает поддержку для использования с инструментами ADO.NET. Приложения, использующие Connector / NET, могут быть написаны на любом поддерживаемом языке .NET.

MySQL Connectors

- MySQL для Visual Studio работает с MySQL Connectors
- MySQL для Visual Studio работает с Connector / NET и Microsoft Visual Studio 2012, 2013, 2015 и 2017. MySQL для Visual Studio обеспечивает доступ к объектам MySQL и данным из Visual Studio. Как пакет Visual Studio, он интегрируется непосредственно в обозреватель сервера, предоставляя возможность создавать новые подключения и работать с объектами базы данных MySQL.
- Connector / ODBC обеспечивает поддержку драйверов для подключения к MySQL с помощью API-интерфейса Open Database Connectivity (ODBC). Поддержка ODBC доступна для платформ Windows, Unix и macOS.
- Connector / Python обеспечивает поддержку драйверов для подключения к MySQL из приложений Python с использованием API, который совместим с API-интерфейсом Python DB версии 2.0. Никаких дополнительных модулей Python или клиентских библиотек MySQL не требуется.
- Connector / NET и Microsoft Visual Studio 2012, 2013, 2015 и 2017. MySQL для Visual Studio обеспечивает доступ к объектам MySQL и данным из Visual Studio. Как пакет Visual Studio, он интегрируется непосредственно в обозреватель сервера, предоставляя возможность создавать новые подключения и работать с объектами базы данных MySQL.

• Connector / ODBC обеспечивает поддержку драйверов для подключения к MySQL с помощью API-интерфейса Open Database Connectivity (ODBC). Поддержка ODBC доступна для платформ Windows, Unix и macOS.
• Connector / Python обеспечивает поддержку драйверов для подключения к MySQL из приложений Python с использованием API, который совместим с API-интерфейсом Python DB версии 2.0. Никаких дополнительных модулей Python или клиентских библиотек MySQL не требуется.

13.2. Сторонние API MySQL

Environment	API	Type	Notes
Ada	GNU Ada MySQL Bindings	libmysqlclient	See MySQL Bindings for GNU Ada
C	C API	libmysqlclient	See Section 28.6, "MySQL C API".
C++	Connector/C++	libmysqlclient	See MySQL Connector/C++ 8.0 Developer Guide.
	MySQL++	libmysqlclient	See MySQL++ website.
	MySQL wrapped	libmysqlclient	See MySQL wrapped.
Cocoa	MySQL-Cocoa	libmysqlclient	Compatible with the Objective-C Cocoa environment. See http://mysql-cocoa.sourceforge.net/
D	MySQL for D	libmysqlclient	See MySQL for D.
Eiffel	Eiffel MySQL	libmysqlclient	See Section 28.12, "MySQL Eiffel Wrapper".
Erlang	erlang-mysql-driver	libmysqlclient	See erlang-mysql-driver.
Haskell	Haskell MySQL Bindings	Native Driver	See Brian O'Sullivan's pure Haskell MySQL bindings.
	hsql-mysql	libmysqlclient	See MySQL driver for Haskell.
Java/JDBC	Connector/J	Native Driver	See MySQL Connector/J 5.1 Developer Guide.
Kaya	MyDB	libmysqlclient	See MyDB.
Lua	LuaSQL	libmysqlclient	See LuaSQL.
.NET/Mono	Connector/NET	Native Driver	See MySQL Connector/NET Developer Guide.
Objective Caml	OBjective Caml MySQL Bindings	libmysqlclient	See MySQL Bindings for Objective Caml.

Octave	Database bindings for GNU Octave	libmysqlclient	See Database bindings for GNU Octave.
ODBC	Connector/ODBC	libmysqlclient	See MySQL Connector/ODBC Developer Guide.
Perl	DBI/DBD::mysql	libmysqlclient	See Section 28.8, "MySQL Perl API".
	Net::MySQL	Native Driver	See Net::MySQL at CPAN.
PHP	mysql, ext/mysql interface (deprecated)	libmysqlclient	See Original MySQL API.
	mysqli, ext/mysqli interface	libmysqlclient	See MySQL Improved Extension.
	PDO_MYSQL	libmysqlclient	See MySQL Functions (PDO_MYSQL).
	PDO mysqlnd	Native Driver	
Python	Connector/Python	Native Driver	See MySQL Connector/Python Developer Guide.
Python	Connector/Python C Extension	libmysqlclient	See MySQL Connector/Python Developer Guide.
	MySQLdb	libmysqlclient	See Section 28.9, "MySQL Python API".
Ruby	MySQL/Ruby	libmysqlclient	Uses libmysqlclient. See Section 28.10.1, "The MySQL/Ruby API".
	Ruby/MySQL	Native Driver	See Section 28.10.2, "The Ruby/MySQL API".
Scheme	Myscsh	libmysqlclient	See Myscsh.
SPL	sql_mysql	libmysqlclient	See sql_mysql for SPL.
Tcl	MySQLtcl	libmysqlclient	See Section 28.11, "MySQL Tcl API".

13.3. MySQL PHP API

PHP предоставляет четыре различных расширения MySQL API:

"MySQL Improved Extension": расшифровывается как « Улучшенное MySQL » ; это расширение доступно с версии PHP 5.0.0. Он предназначен для использования с MySQL 4.1.1 и более поздними версиями. Это расширение полностью поддерживает протокол аутентификации, используемый в MySQL 5.0, а также API подготовленных операторов и множественных

операторов. Кроме того, это расширение предоставляет расширенный объектно-ориентированный интерфейс программирования.

"MySQL Functions (PDO_MYSQL)": это не собственный API, а драйвер MySQL для PDO уровня абстракции базы данных PHP (объекты данных PHP). Драйвер PDO MySQL находится на уровне ниже самого PDO и обеспечивает функциональность, специфичную для MySQL. Это расширение доступно с версии PHP 5.1.0.

Mysql xdevapi: это расширение PECL с именем mysql_xdevapi. Это расширение доступно как расширение PECL с именем mysql_xdevapi. Это расширение предназначено для использования с версиями MySQL до MySQL 4.1. Это расширение не поддерживает улучшенный протокол аутентификации, используемый в MySQL 4.1, и не поддерживает подготовленные операторы или множественные операторы. Чтобы использовать это расширение с MySQL 4.1,

"Original MySQL API": Доступно для версий PHP 4 и 5, это расширение предназначено для использования с версиями MySQL до MySQL 4.1. Это расширение не поддерживает улучшенный протокол аутентификации, используемый в MySQL 4.1, и не поддерживает подготовленные операторы или множественные операторы. Чтобы использовать это расширение с MySQL 4.1,

Обзор драйверов MySQL PHP

В зависимости от версии PHP, есть два или три PHP API для доступа к базе данных MySQL. Пользователи PHP 5 могут выбирать между устаревшим расширением mysql , mysqli или PDO_MySQL . PHP 7 удаляет расширение mysql, оставляя только два последних варианта.

Обзор терминологии

Что такое API?

Интерфейс прикладного программирования, или API, определяет классы, методы, функции и переменные, которые ваше приложение должно будет вызывать для выполнения желаемой задачи. В случае PHP-приложений, которым необходимо взаимодействовать с базами данных, необходимые API обычно предоставляются через расширения PHP.

Что такое API?

Интерфейс прикладного программирования, или API, определяет классы, методы, функции и переменные, которые ваше приложение должно будет вызывать для выполнения желаемой задачи. В случае PHP-приложений, которым необходимо взаимодействовать с базами данных, необходимые API обычно предоставляются через расширения PHP.

Что такое connector?

В документации MySQL термин «*соединитель*» относится к программному обеспечению, которое позволяет вашему приложению подключаться к серверу базы данных MySQL. MySQL предоставляет коннекторы для различных языков, включая PHP.

What is a Driver?

A driver is a piece of software designed to communicate with a specific type of database server. The driver may also call a library, such as the MySQL Client Library or the MySQL Native Driver. These libraries implement the low-level protocol used to communicate with the MySQL database server.

Например, уровень абстракции базы данных PHP Data Objects (PDO) может использовать один из нескольких драйверов, специфичных для базы данных. Одним из доступных драйверов является драйвер PDO MYSQL, который позволяет ему взаимодействовать с сервером MySQL.

What is an Extension?

В документации PHP вы встретите другой термин - *расширение*. Ядро PHP состоит из ядра с необязательными расширениями основных функций. Связанные с MySQL расширения PHP, такие как mysqli расширение и mysqlрасширение, реализованы с использованием инфраструктура расширений PHP.

13.4. Сравнение трех API MySQL

```
<?php
// mysqli
$mysqli = new mysqli("example.com", "user", "password", "database");
$result = $mysqli->query("SELECT 'Hello, dear MySQL user!' AS _message
FROM DUAL"); $row = $result->fetch_assoc();
echo htmlentities($row['_message']);

// PDO $pdo = new PDO('mysql:host=example.com;dbname=database',
'user', 'password'); $statement = $pdo->query("SELECT 'Hello, dear MySQL user!'
AS _message FROM DUAL"); $row = $statement->fetch(PDO::FETCH_ASSOC);
echo htmlentities($row['_message']);

// mysql $c = mysql_connect("example.com", "user", "password");
mysql_select_db("database");
$result = mysql_query("SELECT 'Hello, dear MySQL user!' AS _message
FROM DUAL");
$row = mysql_fetch_assoc($result);
echo htmlentities($row['_message']); ?>
```

Появилось в версии PHP	5.0	5.1	2.0
Работает в PHP 5.x	Да	Да	Да
Работает в PHP 7.x	Да	Да	Нет
Статус разработки	Активный	Активный	Поддержка в 5.x; убрано в 7.x
Жизненный цикл	Активный	Активный	Устаревшее в 5.x; убрано в 7.x
Рекомендовано для новых проектов	Да	Да	Нет
ООП интерфейс	Да	Да	Нет
Процедурный интерфейс	Да	Нет	Да

API поддерживает асинхронные, буферизующие запросы mysqlnd	Да	Нет	Нет
Постоянные(persistent) соединения	Да	Да	Да
API поддерживает кодировки(charset)	Да	Да	Да
API поддерживает подготовленные запросы на стороне сервера	Да	Да	Нет
API поддерживает подготовленные запросы на стороне клиента	Нет	Да	Нет
API поддерживает хранимые процедуры	Да	Большинство	нет
API поддерживает множественные запросы	Да	Да	нет
API поддерживает транзакции	Да	Да	Да
Можно контролировать транзакции посредством SQL	Да	Да	нет
Поддерживает всю функциональность MySQL 5.1+	Да	Большинство	нет

13.5. MySQL PHP API. Примери

```
mysql> CREATE DATABASE mydb; mysql>
CREATE TABLE employees (
id tinyint(4) DEFAULT '0' NOT NULL AUTO_INCREMENT,
first varchar(20),
last varchar(20),
address varchar(255),
position varchar(50),
PRIMARY KEY (id),
UNIQUE id (id));
INSERT INTO employees
VALUES (1,'Bob','Smith','128 Here St, Cityname','Marketing Manager');
INSERT INTO employees VALUES (2,'John','Roberts','45 There St
,Townville','Telephonist');
INSERT INTO employees VALUES (3,'Brad','Johnson','1/34 Nowhere Blvd,
Snowston','Doorman');
```

Выведем эти данные из *базы данных* в HTML-страницу. Для общения с MySQL из *PHP* понадобятся следующие функции.

int *mysql_connect*(string hostname, string username, string password);

- создать соединение с MySQL.

Параметры:

Hostname – *имя хоста*, на котором находится *база данных*.
Username – *имя пользователя*.
Password – *пароль* пользователя.
Функция возвращает *параметр* типа int, который больше 0, если соединение прошло успешно, и равен 0 в противном случае.
int *mysql_select_db*(string database_name, int link_identifier); - выбрать базу данных для работы.
Параметры:
Database_name – имя *базы данных*.
link_identifier – ID соединения, которое получено в функции *mysql_connect*. (*параметр* необязательный, если он не указывается, то используется ID от последнего вызова *mysql_connect*)
Функция возвращает *значение* true или false
int *mysql_query*(string query, int link_identifier); - функция выполняет запрос к базе данных.
Параметры:
query – строка, содержащая *запрос*
link_identifier – см. предыдущую функцию.
Функция возвращает *ID* результата или 0, если произошла ошибка.
int *mysql_result*(int result, int i, column); - функция возвращает значение поля в столбце column и в строке i.
int *mysql_close*(int link_identifier); - функция закрывает соединение с MySQL.
Параметры:

```
<html>
<body>
<?php
$db = mysql_connect("localhost", "root");
mysql_select_db("mydb",$db);
$result = mysql_query("SELECT * FROM employees",$db);
printf("First Name: %s<br>\n", mysql_result($result,0,"first"));
printf("Last Name: %s<br>\n", mysql_result($result,0,"last"));
printf("Address: %s<br>\n",
mysql_result($result,0,"address"));
printf("Position: %s<br>\n",
mysql_result($result,0,"position"));
mysql_close($db);
?>
</body>
</html>
<html>
<body>
<?php
```



```
$db = mysql_connect("localhost", "root"); mysql_select_db("
$result = mysql_query("SELECT * FROM employees",$db);
echo "<table border=1>\n";
while ($myrow = mysql_fetch_row($result))
{printf("<tr><td>%s</td><td>%s</td><td>%s</td></tr>\n", $myrow[1], $myrow[2],
$myrow[3]); }
echo "</table>\n";
?>
</body>
</html>
<html>
<body>
<?php
$db = mysql_connect("localhost", "root");
mysql_select_db("mydb",$db);
$result = mysql_query("SELECT * FROM employees",$db);
if ($myrow = mysql_fetch_array($result))
{ echo "<table border=1>\n";
echo "<tr><td>Name</td><td>Position</td></tr>\n";
Do {printf("<tr><td>%s %s</td><td>%s</tr>\n", $myrow["first"],
$myrow["last"], $myrow["address"]); }
while ($myrow = mysql_fetch_array($result));
echo "</table>\n"; }
Else
{ echo "Sorry, no records were found!"; }
?>
</body>
</html>
```

14. XML и база данных (MySQL)

14.1. Что такое XML?

XML (eXtensible Markup Language) - расширяемый язык разметки. Основное внимание в XML сосредоточено на данных. В XML структура разметка данных и представление данных строго разделены.

Основные причины создания XML:

1. попытка предоставить мощные средства форматирования и структурирования данных всем желающим;
2. необходимость в стабильной реализации языка структурирования документов, для которого легко было бы создавать вспомогательные инструменты, доступные для обычных пользователей.

XML является метаязыком - специальным языком, на котором можно составить полное описание класса других языков, на которых, в свою очередь, составляются документы. Он представляет собой набор правил, позволяющих создавать приложения и подмножества данных, уникальные для их задач.

Каждый документ XML должен начинаться с пролога, указывающего версию используемого языка XML и метод кодировки, например:

```
<?xml version="1.0" encoding="WINDOWS-1251"?>
<? xml version="1.0" encoding="WINDOWS-1251" ?>
<memo>
<recipient> Илдаров М. </recipient>
<message>Командировка отменяется. Подготовьте материалы.
</message>
</memo>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

Пример 2

```
<!-- Прайс-лист фруктов -->
<dl>
<!-- Тип фруктов -->
<dt>Яблоки</dt>
<!-- Цена -->
<dd>$1</dd>
<!-- Тип фруктов -->
<dt>Мандарины</dt>
<!-- Цена -->
```



```
<dd>$2</dd>
</dl>
```

Пример 3

```
<Прайс_лист_фруктов>
<Фрукт>Яблоки</Фрукт>
<Цена>$1</Цена>
<Фрукт>Мандарины</Фрукт>
<Цена>$2</Цена>
</Прайс_лист_фруктов>
```

14.2. Функции XML

название	Описание
ExtractValue()	Извлечение значения из строки XML с использованием нотации XPath
UpdateXML()	Вернуть замененный фрагмент XML

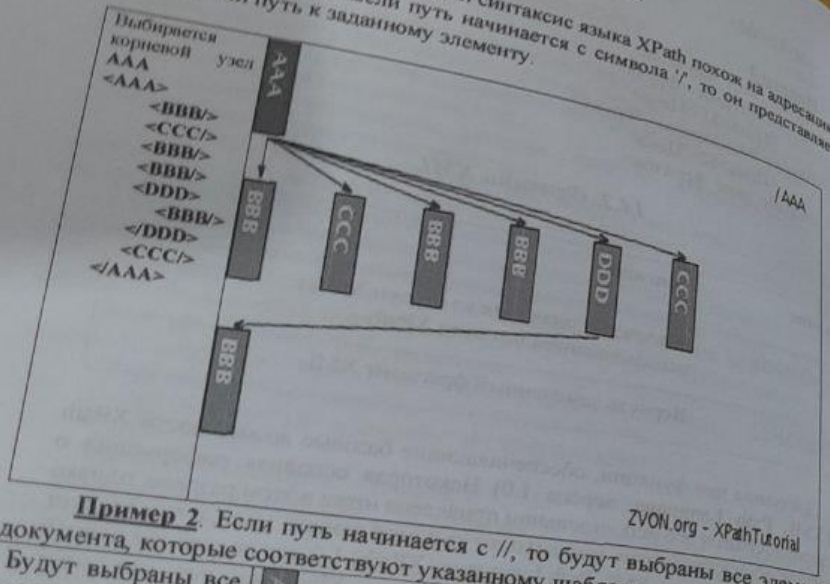
Доступны две функции, обеспечивающие базовые возможности XPath 1.0 (XML Path Language, версия 1.0). Некоторая основная информация о синтаксисе XPath и его использовании приведена ниже в этом разделе; однако подробное обсуждение этих тем выходит за рамки данной темы, и вам следует обратиться к стандарту XML Path Language (XPath) 1.0 для получения полной информации. Полезным ресурсом для новичков в XPath или тех, кто хочет освежиться в основах, является Учебное пособие по XPath от Zvon.org, которое доступно на нескольких языках.

Выражения XPath, используемые с этими функциями, поддерживают пользовательские переменные и локальные хранимые программные переменные. Пользовательские переменные слабо проверяются; Строго проверяются переменные, локальные для хранимых процедур.

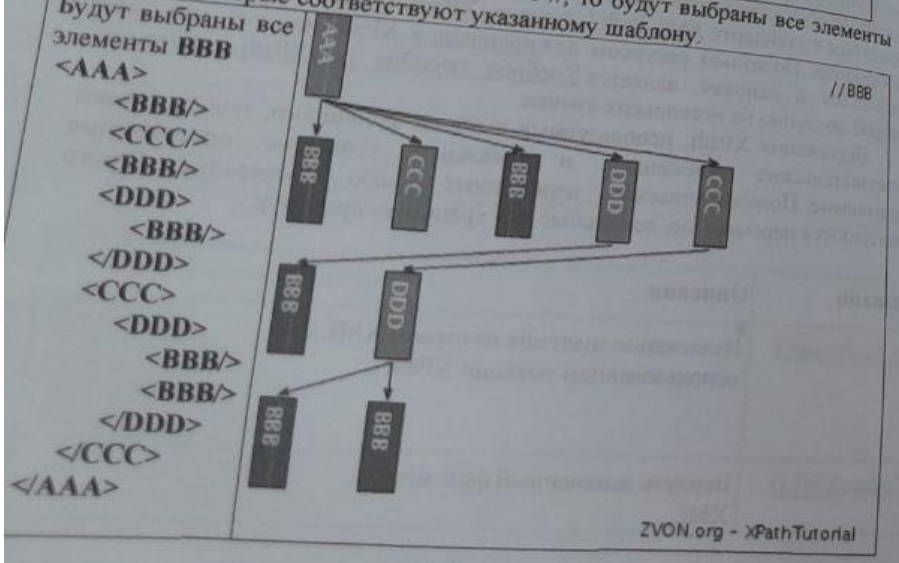
названи е	Описани е
ExtractValue()	Извлечение значения из строки XML с использованием нотации XPath
UpdateXML()	Вернуть замененный фрагмент XML

14.3. Примеры Xpath

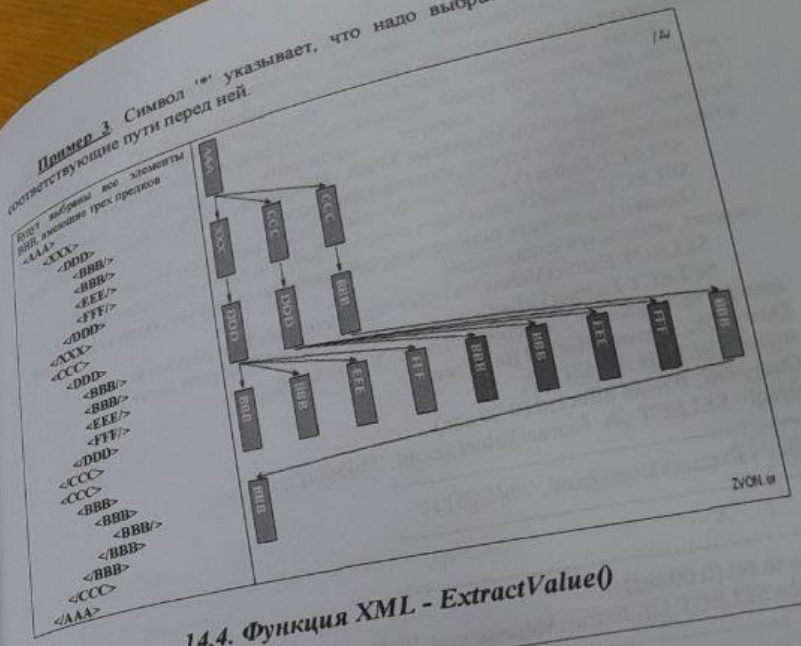
Пример 1 (/AAA). Базовый синтаксис языка XPath похож на адресацию в файловой системе. Если путь начинается с символа '/', то он представляет абсолютный путь к заданному элементу.



Пример 2. Если путь начинается с //, то будут выбраны все элементы документа, которые соответствуют указанному шаблону.



Пример 3. Символ '*' указывает, что надо выбрать все элементы, соответствующие пути перед ней.



14.4. Функция XML - ExtractValue()

название	Описание
ExtractValue()	Извлечение значения из строки XML с использованием нотации XPath

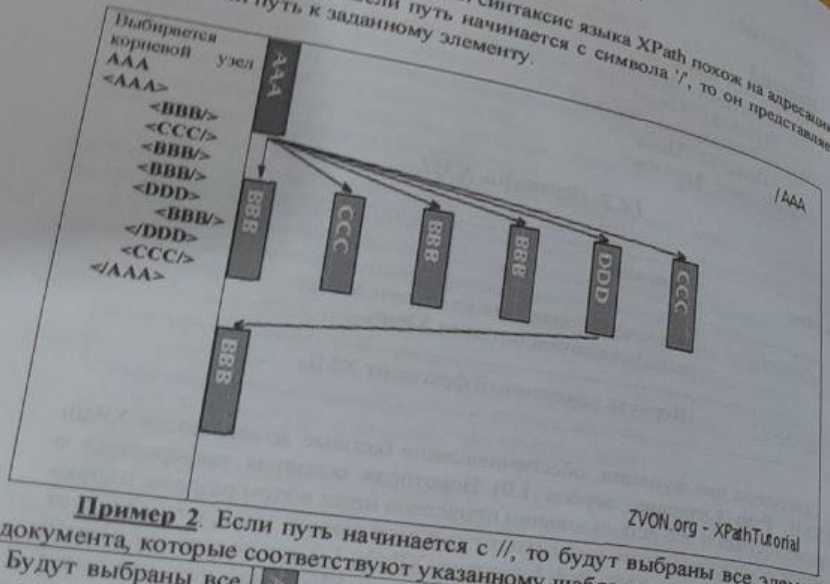
ExtractValue(*xml_frag, xpath_expr*)

ExtractValue()принимает два строковых аргумента, фрагмент разметки XML *xml_fragи* выражение XPath *xpath_expr*(также известное как локатор); он возвращает текст (CDATA) первого текстового узла, который является дочерним элементом элемента или элементов, соответствующих выражению XPath.

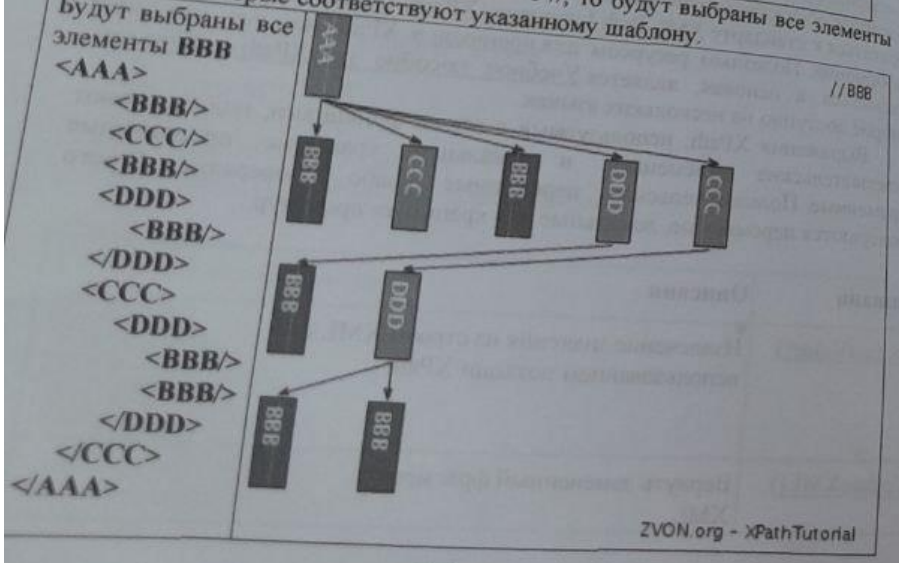
Пользовательские переменные (слабая проверяется). Переменные, использующие синтаксис (то есть пользовательские переменные), не проверяются. Сервер не выдает предупреждений или ошибок, если переменная имеет неправильный тип или ей ранее не было присвоено

14.3. Примеры Xpath

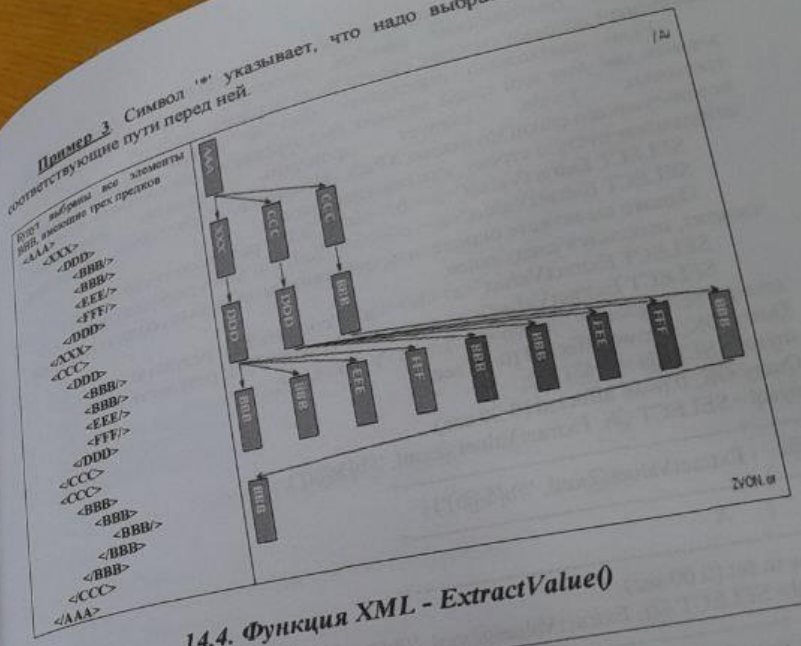
Пример 1 (/AAA). Базовый синтаксис языка XPath похож на адресацию в файловой системе. Если путь начинается с символа '/', то он представляет абсолютный путь к заданному элементу.



Пример 2. Если путь начинается с //, то будут выбраны все элементы документа, которые соответствуют указанному шаблону.



Пример 3. Символ '*' указывает, что надо выбрать все элементы, соответствующие пути перед ней.



14.4. Функция XML - ExtractValue()

название	Описание
ExtractValue()	Извлечение значения из строки XML с использованием нотации XPath

ExtractValue(*xml_frag, xpath_expr*)

ExtractValue()принимает два строковых аргумента, фрагмент разметки XML *xml_fragи* выражение XPath *xpath_expr*(также известное как локатор); он возвращает текст (CDATA) первого текстового узла, который является дочерним элементом элемента или элементов, соответствующих выражению XPath.

Пользовательские переменные (слабая проверяется). Переменные, использующие синтаксис (то есть пользовательские переменные), не проверяются. Сервер не выдает предупреждений или ошибок, если переменная имеет неправильный тип или ей ранее не было присвоено

значение. Это также означает, что пользователь несет полную ответственность за любые типографские ошибки, поскольку предупреждения, если (например) используется там, где это было задумано, не будут. Если необходимо определить, был ли найден соответствующий элемент *xml_frag* или такой элемент был найден, но не содержал дочерних текстовых узлов, следует проверить результат выражения; используюшего count()функцию XPath . Например, оба этих оператора возвращают пустую строку, как показано здесь:

```
SELECT ExtractValue('<a><b/></a>', '/a/b'); результат пусто
SELECT ExtractValue('<a><c/></a>', '/a/b'); и здесь результат пусто
```

Однако вы можете определить, действительно ли был соответствующий элемент, используя следующее:

```
SELECT ExtractValue('<a><b/></a>', 'count(/a/b)'); результат 1
SELECT ExtractValue('<a><c/></a>', 'count(/a/b)'); результат 0
mysql> SET @xml = '<a><b>X</b><b>Y</b></a>';
Query OK, 0 rows affected (0.00 sec)
mysql> SET @i =1, @j = 2;
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT @i, ExtractValue(@xml, '//b[$@i]');
+------+-------------------------------------+
| @i   | ExtractValue(@xml, '//b[$@i]') |
+------+-------------------------------------+
| 1    | X                                   |
+------+-------------------------------------+
1 row in set (0.00 sec)
mysql> SELECT @j, ExtractValue(@xml, '//b[$@j]');
+------+-------------------------------------+
| @j   | ExtractValue(@xml, '//b[$@j]') |
+------+-------------------------------------+
| 2    | Y                                   |
+------+-------------------------------------+
1 row in set (0.00 sec)
mysql> SELECT @k, ExtractValue(@xml, '//b[$@k]');
+------+-----------------------------+
| @k | ExtractValue(@xml, '//b[$@k]') |
+------+-----------------------------+
| NULL |                             |
+------+-----------------------------+
row in set (0.00 sec)
```

Переменные в хранимых программах (строгая проверка). Переменные, использующие синтаксис, могут быть объявлены и использованы с этими функциями, когда они вызываются внутри хранимых программ. Такие переменные являются локальными для хранимой

программы, в которой они определены, и строго проверяются на тип и значение. *$variable_name*

Пример:

```
mysql> DELIMITER |
mysql> CREATE PROCEDURE myproc ()
BEGIN
  DECLARE i INT DEFAULT 1;
  DECLARE xml VARCHAR(25) DEFAULT
'<a>X</a><a>Y</a><a>Z</a>';
  WHILE i < 4 DO
  SELECT xml, i, ExtractValue(xml, '//a[$i]');
  SET i = i+1;
  END WHILE;
END
| Query OK, 0 rows affected (0.01 sec)
mysql> DELIMITER ;
mysql> CALL myproc();
```

14.5. Оператор LOAD XML

```
LOAD XML
[LOW_PRIORITY | CONCURRENT] [LOCAL]
INFILE 'file_name'
[REPLACE | IGNORE]
INTO TABLE [db_name.]tbl_name
[CHARACTER SET charset_name]
[ROWS IDENTIFIED BY '<tagname>']
[IGNORE number {LINES | ROWS}]
[(field_name_or_user_var
    [, field_name_or_user_var] ...)]
[SET col_name={expr | DEFAULT},
    [, col_name={expr | DEFAULT}] ...]
```

14.6. Оператор UPDATEXML

UpdateXML(*xml_target, xpath_expr, new_xml*)

Эта функция заменяет одну часть заданного фрагмента разметки *xml_target*XML новым фрагментом XML *new_xml*, а затем возвращает измененный XML. Заменяемая часть *xml_target* соответствует выражению XPath, *xpath_expr* предоставленному пользователем.

Если совпадение выражений *xpath_expr* не найдено или найдено несколько совпадений, функция возвращает исходный *xml_target* фрагмент XML. Все три аргумента должны быть строками.

```
mysql> SELECT
UpdateXML(xml_target, xpath_expr, new_xml)
```

Эта функция заменяет одну часть заданного фрагмента XML разметки *xml_target* XML новым фрагментом *new_xml*, а затем возвращает измененный XML. Заменяемая часть *xml_target* соответствует выражению XPath, *xpath_expr* предоставленному пользователем.

Если совпадение выражений *xpath_expr* не найдено или найдено несколько совпадений, функция возвращает исходный *xml_target* фрагмент XML. Все три аргумента должны быть строками.

```
mysql> SELECT
```


15. СИСТЕМЫ РАСПРЕДЕЛЕННОЙ ОБРАБОТКИ ДАННЫХ. ПЕРЕХОД К РАСПРЕДЕЛЕННОЙ ОБРАБОТКЕ ДАННЫХ. АРХИТЕКТУРА СОВРЕМЕННЫХ РАСПРЕДЕЛЕННЫХ СУБД.

Введение.

Основной предпосылкой разработки систем, использующих базы данных, является стремление объединить все обрабатываемые в организации данные в единое целое и обеспечить к ним контролируемый доступ. Хотя интеграция и предоставление контролируемого доступа могут способствовать централизации, последняя не является самоцелью. На практике создание компьютерных сетей приводит к децентрализации обработки данных. Децентрализованный подход, по сути, отражает организационную структуру многих компаний, логически состоящих из отдельных подразделений, отделов, проектных групп и т.п., которые физически распределены по разным офисам, отделениям, предприятиям или филиалам, причем каждая отдельная производственная единица имеет дело с собственным набором обрабатываемых данных. Разработка распределенных баз данных, отражающих организационные структуры предприятий, позволяет сделать общедоступными данные, поддерживаемые каждым из существующих подразделений, обеспечив при этом их хранение именно в тех местах, где они чаще всего используются. Подобный подход расширяет возможности совместного использования информации, одновременно повышая эффективность доступа к ней.

Распределенные системы призваны решить проблему информационных островов. Если на предприятии имеется несколько баз данных, их иногда рассматривают как некие разрозненные территории, представляющие собой отдельные и труднодоступные для многих места, подобные удаленным друг от друга островам. Данное положение может являться следствием географической разобщенности, несовместимости используемой компьютерной архитектуры, несовместимости используемых протоколов связи и т.д. Подобное положение дел способна изменить интеграция отдельных баз данных в одно логическое целое.

15.1. *Основные понятия.*

Системы дистрибутивных баз данных состоят из набора узлов, связанных вместе коммуникационной сетью, в которой:

1. каждый узел обладает своими собственными системами баз данных;
2. узлы работают согласованно, поэтому пользователь может получить доступ к данным на любом узле сети, как будто все данные находятся на его собственном узле.

Из этого следует, что так называемая "распределенная база данных" на самом деле является типом виртуального объекта, части которого физически сохраняются в ряду отдаленных "реальных" баз данных на удаленных узлах (фактически, это логическая единица всех этих реальных баз данных).

Распределенная база данных. Набор логически связанных между собой совокупностей разделяемых данных (и их описаний), которые физически распределены в некоторой компьютерной сети.

Из этого вытекает следующее определение распределенной СУБД.

Распределенная СУБД. Программный комплекс, предназначенный для управления распределенными базами данных и обеспечивающий прозрачный доступ; пользователей к распределенной информации.

Распределенная система управления базой данных (распределенная СУБД) состоит из единой логической базы данных, разделенной на некоторое количество фрагментов. Каждый фрагмент базы данных сохраняется на одном или нескольких компьютерах, работающих под управлением отдельных СУБД и соединенных между собой сетью связи. Любой узел способен независимо обрабатывать запросы пользователей, требующие доступа к локально сохраняемым данным (т.е. каждый узел обладает определенной степенью автономности), а также способен обрабатывать данные, сохраняемые на других компьютерах сети.

Пользователи взаимодействуют с распределенной базой данных через приложения. Приложения могут подразделяться на не требующие доступа к данным на других узлах (локальные приложения) и требующие подобного доступа (глобальные приложения). В распределенной СУБД должно существовать хотя бы одно глобальное приложение, поэтому любая такая СУБД должна иметь следующие характеристики.

- Имеется набор логически связанных разделяемых данных.
- Сохраняемые данные разбиты на некоторое количество фрагментов.
- Может быть предусмотрена репликация фрагментов данных.
- Фрагменты и их копии распределяются по разным узлам.
- Узлы связаны между собой сетевыми соединениями.
- Доступ к данным на каждом узле происходит под управлением СУБД.
- СУБД на каждом узле способна поддерживать автономную работу локальных приложений.
- СУБД каждого узла поддерживает хотя бы одно глобальное приложение.

Но нет необходимости в том, чтобы на каждом из узлов системы существовала своя собственная локальная база данных, что и показано на примере топологии распределенной СУБД, представленной на рис. 1.

Из определения СУБД следует, что она должна сделать само это определение данных прозрачным (незаметным) для конечного пользователя. Другими словами, от пользователей должен быть полностью

скрыт тот факт, что распределенная база данных состоит из нескольких фрагментов, которые могут размещаться на различных компьютерах и для которых, возможно, даже организована репликация данных. Цель обеспечения прозрачности состоит в том, чтобы распределенная система внешне выглядела как централизованная. Иногда это требование называют основным принципом создания распределенных СУБД. Данный принцип требует предоставления конечному пользователю широкого набора функциональных возможностей, но, к сожалению, одновременно ставит перед программным обеспечением распределенной СУБД множество дополнительных задач.

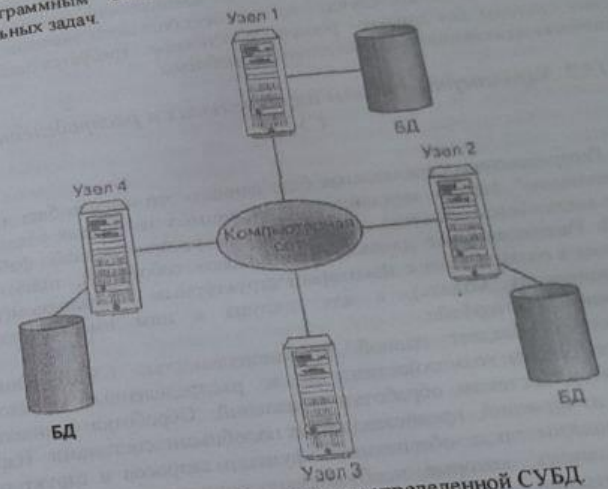


Рисунок 1. Топология распределенной СУБД

В распределенной базе данных каждый узел обладает своими собственными базами данных, собственными локальными пользователями, собственной СУБД и программным обеспечением для управления транзакциями (включая собственное программное обеспечение для блокирования, регистрации, восстановления и т.д.), а также своим собственным локальным диспетчером передачи данных. В частности, пользователь может на собственном локальном узле выполнять операции с данными так, как будто этот узел вовсе не является частью распределенной системы.

Чаще всего предполагается, что узлы физически распределены (а возможно, и географически, как показано на рис. 1), хотя в

действительности достаточно того, чтобы они были распределены логически. Два"узла" могут даже сосуществовать на одной и той же машине(в особенностей на начальном этапе тестирования). Главная цель создания распределенных систем со временем изменялась. В ранних исследованиях в основном предполагалась географическая распределенность, но в большинстве первых коммерческих реализаций предполагалось локальное распределение, когда несколько "узлов" размещалось в одном здании и соединялось с помощью локальной сети(ЛВС). Однако позже стремительное распространение глобальных сетей(ГВС) снова пробудило интерес к возможности географического распределения. В любом случае это не имеет большого значения с точки зрения системы баз данных– решать, в основном, требуется одни и те же технические(связанные с базами данных) проблемы.

15.2. Характерные черты параллельных и распределенных СУБД.

1. Распределенная/параллельная база данных- это именно база данных, а не "коллекция" файлов, индивидуально хранимых на разных узлах сети. В этом заключается разница междуDDB и распределенной файловой системой. Распределенные данные представляют собойDDB, только если они связаны в соответствии с некоторым структурным формализмом(таким как реляционная модель), а для доступа к ним имеется единый высокоуровневый интерфейс.

2. Система обладает полной функциональностью СУБД. Она не сводится по своим возможностям ни к распределенным файловым системам, ни к системам обработки транзакций. Обработка транзакций- только одна из функций, предоставляемых подобными системами. Наряду с этим они должны также обеспечивать функции запросов и структурной организации данных, которые необязательно поддерживаются системами обработки транзакций.

3. Распределение(включая фрагментацию и репликацию) данных по множеству узлов невидимо для пользователей. Это свойство называется прозрачностью. Технология распределенных/параллельных баз данных распространяет основополагающую для управления базами данных концепцию независимости данных на среду, где данные распределены и реплицированы по множеству компьютеров, связанных сетью. Это обеспечивается за счет нескольких видов прозрачности: прозрачность сети(следовательно, прозрачность распределения), прозрачность репликации и прозрачность фрагментации. Прозрачность доступа означает, что пользователи имеют дело с единым логическим образом базы данных и осуществляют доступ к распределенным данным точно так же, как если они хранились централизованно. В идеале полная прозрачность

подразумевает наличие языка запросов к распределенной СУБД, не отличающегося от языка для централизованной СУБД.

15.3. Преимущества

Зачем нужны распределенные базы данных? Основная причина заключается в том, что предприятия обычно уже распределены. Распределены по крайней мере логически, т.е разделены на подразделения, отделы, рабочие группы и т.д. Очень часто они распределены и физически, т.е. разделены на заводы, фабрики, лаборатории и т.д. Из этого следует, что данные также обычно распределены, поскольку каждая организационная единица создает и обрабатывает собственные данные, относящиеся к деятельности этой единицы. Таким образом, информация предприятия разбивается на части, которые иногда называют островами информации. А распределенная система обеспечивает мосты для их соединения в единое целое. Иначе говоря, распределенная система позволяет структуре базы данных отображать структуру предприятия— локальные данные могут храниться локально, в соответствии с логической принадлежностью, тогда как к удаленным данным доступ может осуществляться по мере необходимости.

Для того чтобы лучше в этом разобраться, приведем пример. Рассмотрим еще раз рис. 1. Для простоты предположим, что есть только два узла, Лос-Анджелес и Сан-Франциско, и что наша система– это банковская система. Данные о счетах Лос-Анджелеса хранятся в Лос-Анджелесе, а данные о счетах Сан-Франциско – в Сан-Франциско.

Преимущества подобной распределенной системы очевидны: эффективность обработки(данные хранятся в том месте, где доступ к ним требуется наиболее часто) плюс расширенные возможности доступа(при необходимости с помощью коммуникационной сети из Сан-Франциско можно получить доступ к счетам Лос-Анджелеса и наоборот).

Пожалуй, наиболее важным преимуществом распределенных систем является, как уже было отмечено, отражение ими структуры предприятия. Конечно, существует мно-жество других преимуществ, которые будут обсуждаться ниже в этой главе вместе с со-ответствующими аспектами. Однако следует отметить, что подобным системам свойствен и ряд недостатков, наиболее существенным из которых является повышенная сложность распределенных систем, по крайней мере, с технической точки зрения. В идеальном случае, конечно, эта сложность должна быть проблемой реализации, а не проблемой пользователя, но вполне возможно, что на практике некоторые ее аспекты все-таки будут видны конечным пользователям. Для того чтобы скрыть от пользователя сложность систем, требуется весьма тщательная ее проработка.

15.4. Однородные и разнородные распределенные СУБД

Распределенные СУБД подразделяются на однородные и разнородные. В однородных системах все узлы используют один и тот же тип СУБД. В разнородных системах на узлах могут функционировать различные типы СУБД, использующие разные модели данных, т.е. разнородная система может включать узлы с реляционными, сетевыми, иерархическими или объектно-ориентированными СУБД.

Однородные системы значительно проще проектировать и сопровождать. Кроме того, подобный подход позволяет поэтапно наращивать размеры системы, последовательно добавляя новые узлы к уже существующей распределенной системе. Дополнительно появляется возможность повышать производительность системы за счет организации на различных узлах параллельной обработки информации.

Разнородные системы обычно возникают в тех случаях, когда независимые узлы, уже эксплуатирующие свои собственные системы с базами данных, со временем интегрируются во вновь создаваемую распределенную систему. В разнородных системах для организации взаимодействия между различными типами СУБД требуется обеспечить преобразование передаваемых сообщений. Для обеспечения прозрачности в отношении типа используемой СУБД пользователи каждого из узлов должны иметь возможность формулировать интересующие их запросы на языке той СУБД, которая используется на их локальном узле. Система должна взять на себя поиск требуемых данных и выполнение всех необходимых преобразований передаваемых сообщений. В общем случае данные могут быть затребованы с другого узла, который характеризуется следующими особенностями:

- иной тип используемого оборудования;
- иной тип используемой СУБД;
- иной тип применяемых оборудования и СУБД.

Если используется иной тип оборудования, но на узлах применяются такие же СУБД, методы выполнения преобразований вполне очевидны и включают замену кодов и изменение длины машинного слова. Если используемых на узлах СУБД различны, процедура преобразования усложняется тем, что необходимо преобразовывать структуры данных одной модели данных в эквивалентные структуры данных другой модели данных. Например, отношения в реляционной модели данных должны быть преобразованы в записи и таборы, характерные для сетевой модели данных. Кроме того, приходится транслировать текст запросов с одного языка на другой(например, запросы с оператором SELECT языкаSQL может потребоваться преобразовать в запросы с операторамиFIND и GET языка манипулирования данными сетевой СУБД). Если отличаются и тип используемого оборудования, и тип программного обеспечения,

потребуется выполнять оба вида трансляции. Все изложенное выше чрезвычайно усложняет обработку данных в разнородных распределенных СУБД.

Дополнительные сложности возникают при попытках выработки единой концептуальной схемы, создаваемой путем интеграции отдельных локальных концептуальных схем. При наличии семантической неоднородности интеграция локальных моделей данных становится чрезвычайно трудной задачей. Например, атрибуты, имеющие в разных схемах одно и то же имя, на деле могут представлять совершенно различные понятия. Аналогично, атрибуты с разными именами фактически могут представлять одну и ту же характеристику.

Типичное решение, применяемое в некоторых реляционных системах, состоит в том, что отдельные части разнородных распределенных систем должны использовать шлюзы, предназначенные для преобразования языка и модели данных каждого из используемых типов СУБД в язык и модель данных реляционной системы. Однако подходу с применением шлюзов свойственны некоторые серьезные ограничения. Во-первых, шлюзы не позволяют организовать систему управления транзакциями даже для отдельных пар систем. Другими словами, шлюз между двумя системами представляет собой не более чем транслятор запросов. Например, шлюзы не позволяют системе координировать управление параллельным выполнением и процедурами восстановления транзакций, включающих обновление данных в обеих базах. Во-вторых, использование шлюзов позволяет решить лишь задачу трансляции запросов с языка одной СУБД на язык другой. Поэтому они, как правило, не позволяют решить проблему создания однородной структуры и устранить различия между представлениями данных в различных схемах.

15.5. Функции распределенных СУБД

Следует ожидать, что типичная распределенная СУБД должна обеспечивать, по крайней мере, тот же набор функциональных возможностей, который был определен для централизованных СУБД. Кроме того, распределенная СУБД должна иметь следующий набор функциональных возможностей.

- Расширенные службы установки соединений должны обеспечивать доступ к удаленным узлам и позволять передавать запросы и данные между узлами, входящими в сеть.
- Расширенные средства ведения каталога, позволяющие сохранять сведения о распределении данных в сети.
- Средства обработки распределенных запросов, включая механизмы оптимизации запросов и организации удаленного доступа к данным.

- Расширенные функции управления защитой, позволяющие обеспечить соблюдение правил авторизации и прав доступа к распределенным данным.
- Расширенные функции управления параллельным выполнением, позволяющие поддерживать целостность копируемых данных.
- Расширенные функции восстановления, учитывающие вероятность отказов в работе отдельных узлов и отказов линий связи.

15.6. Архитектура распределенных СУБД

Трехуровневая архитектура ANSI-SPARC для СУБД, представляет собой типовое решение для централизованных СУБД. Однако распределенные СУБД имеют множество отличий, которые весьма сложно отразить в некотором эквивалентном архитектурном решении, приемлемом для большинства случаев. Но было бы полезно найти какие-либо рекомендуемые решения, учитывающие особенности работы с распределенными данными. Один из примеров рекомендуемой архитектуры распределенной СУБД представлен на рис. 2.

Он включает следующие компоненты:

- набор глобальных внешних схем;
- глобальная концептуальная схема;
- схема фрагментации и распределения;
- набор схем для каждой локальной СУБД, отвечающий требованиям трех-уровневой архитектуры ANSI-SPARC.

Соединительные линии на схеме представляют преобразования, выполняемые при переходе между схемами различных типов. Б зависимости от поддерживаемого уровня прозрачности некоторые из уровней рекомендуемой архитектуры могут отсутствовать.

Глобальная концептуальная схема— это логическое описание всей базы данных, представляющее ее так, как будто она не является распределенной. Этот уровень распределенной СУБД соответствует концептуальному уровню архитектурыANSI-SPARC и содержит определения сущностей, связей, требований защиты и ограниченийподдержки целостности информации. Он обеспечивает физическую независимость данных от распределенной среды. Логическую зависимость данных обеспечивают глобальные внешние схемы.

Схема фрагментации описывает, как данные должны логически определяться по разделам. Схема размещения показывает, где положены имеющиеся данные с учетом необходимости репликации.

Локальные схемы. Каждая локальная СУБД имеет свой собственный о схем. Локальные концептуальная и внутренняя схемы полностью ветствуют эквивалентным уровням архитектурыANSI-SPARC,

Локальная схема отображения используется для отображения фрагментов в схеме размещения во внешние объекты локальной базы данных. Эти элементы зависят от типа используемой СУБД и служат осиовой для создания разнородных распределенных СУБД.

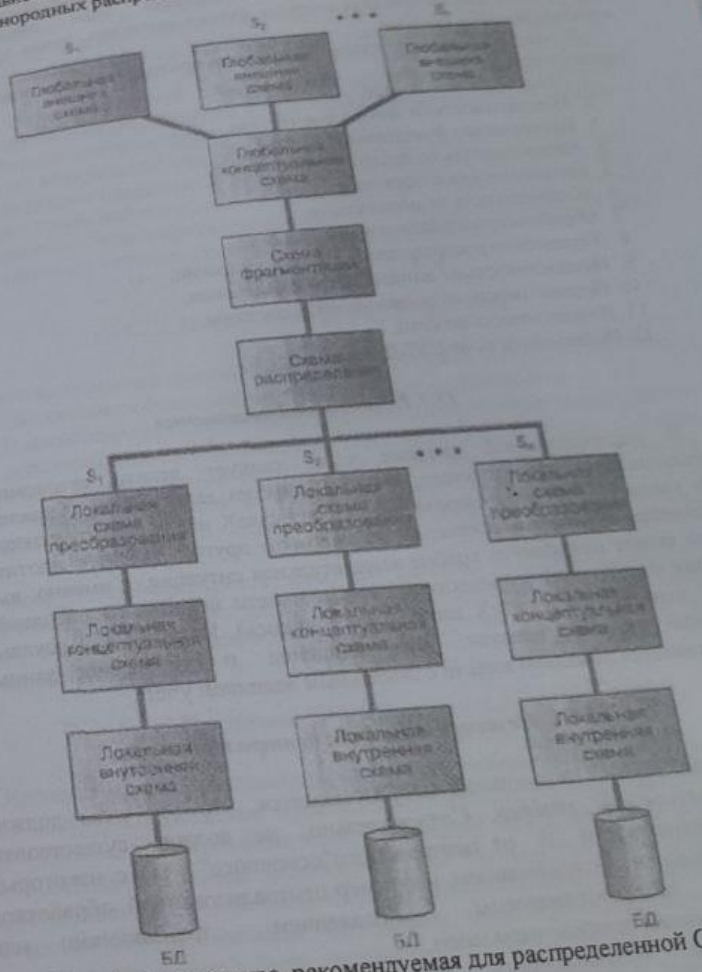


Рис. 2. Архитектура, рекомендуемая для распределенной СУБД

15.7. Принципы функционирования распределенной БД

Теперь, после краткого введения, можно привести формулировку фундаментального принципа распределенной базы данных. Для пользователя распределенная система должна выглядеть точно так же, как нераспределенная система.

Изложенный фундаментальный принцип приводит к следующему набору правил и целей:

1. Локальная автономия;
2. Независимость от центрального узла;
3. Непрерывное функционирование;
4. Независимость от расположения
5. Независимость от фрагментации;
6. Независимость от репликации;
7. Обработка распределенных запросов;
8. Управление распределенными транзакциями;
9. Независимость от аппаратного обеспечения;
10. Независимость от операционной системы;
11. Независимость от сети;
12. Независимость от СУБД.

15.7.1. Локальная автономия

В распределенной системе узлы следует делать автономными. Локальная автономия означает, что операции на данном узле управляются этим узлом, т.е. функционирование любого узлаX не зависит от успешного выполнения некоторых операций на каком-то другом узлеY (в противном случае может возникнуть крайне нежелательная ситуация, а именно: выход из строя узлаY может привести к невозможности исполнения операций на узлеX, даже если с узломX ничего не случилось). Из принципа локальной автономии также следует, что владение и управление данными осуществляется локально вместе с локальным ведением учета.

15.7.2. Независимость от центрального узла

Под локальной автономией подразумевается, что все узлы должны рассматриваться как равные. Следовательно, не должно существовать никакой зависимости и от центрального"основного" узла с некоторым централизованным обслуживанием, например централизованной обработкой запросов, централизованным управлением транзакциями или централизованным присвоением имен.

Локальная независимость предполагает, что все узлы в распределенной системе должны рассматриваться как равные. Поэтому, в

частности, не должно быть никаких обращений к"центральному" или"главному" узлу с целью получения некоторого централизованного сервиса. Не должно быть, например, централизованной обработки запросов, централизованного управления транзакциями или централизованной службы присвоения имен, поскольку в таких случаях система в целом будет зависимой от центрального узла. Таким образом, вторая цель на самом деле является следствием первой цели— если первая цель достигнута, то вторая цель также заведомо достигнута. Но достижение цели"Отсутствие опоры на центральный узел" полезно само по себе, даже если полная локальная независимость узлов не будет достигнута. Поэтому отдельная формулировка данной цели также важна.

Опора на центральный узел является нежелательной по крайней мере по двум следующим причинам. Во-первых, такой центральный узел, скорее всего, станет узким местом системы, и, во-вторых(что еще хуже), система станет уязвимой— если работа центрального узла будет нарушена, то вся система выйдет из строя(проблема"единого источника отказа").

15.7.3. Непрерывное функционирование

Одним из основных преимуществ распределенных систем является то, что они обеспечивают более высокую надежность и доступность.

1. Надежность(вероятность того, что система исправна и работает в любой заданный момент) повышается благодаря работе распределенных систем не по принципу"все или ничего", а в постоянном режиме; т.е. работа системы продолжается, хотя и на более низком уровне, даже в случае неисправности некоторого отдельного компонента, например отдельного узла.

2. Доступность(вероятность того, что система исправна и работает в течение некоторого промежутка времени) повышается частично по той же причине, а частично благодаря возможности репликации данных(подробнее это описывается ниже).

15.7.4. Независимость от расположения

Основная идея независимости от расположения, или так называемой прозрачности расположения, проста. Пользователи не должны знать, где именно данные хранятся физически и должны поступать так(по крайней мере, с логической точки зрения), как если бы все данные хранились на их собственном локальном узле. Благодаря независимости от расположения упрощаются пользовательские программы и терминальные операции. В частности, данные могут быть перенесены с одного узла на другой, и это не должно потребовать внесения каких-либо изменений в использующие

их программы или действия пользователей. Такая переносимость желательна, поскольку она позволяет перемещать данные в сети в соответствии с изменяющимися требованиями к эффективности работы системы.

15.7.5 Независимость от фрагментации

В системе поддерживается фрагментация данных, если некое хранимое отношение в целях физического хранения можно разделить на части, или"фрагменты". Фрагментация желательна для повышения производительности системы, поскольку данные лучше хранить в том месте, где они наиболее часто используются. При такой организации многие операции будут чисто локальными, что снизит нагрузку на сеть.

Существует два основных типа фрагментации– горизонтальная и вертикальная, которые связаны с реляционными операциями выборки и проекции соответственно. Иначе говоря, фрагментом может быть любое произвольное подчиненное отношение, которое можно вывести из исходного отношения с помощью операций выборки и проекции. При этом следует учесть приведенные ниже допущения.

Предполагается без утраты общности, что все фрагменты данного отношения независимы, т.е. ни один из фрагментов не может быть выведен из других фрагментов либо иметь выборку или проекцию, которая может быть выведена из других фрагментов. (Если есть необходимость сохранить одну и ту же информацию в нескольких разных местах, для этого следует использовать механизм репликации системы.). Проекции не должны допускать потерю информации.

15.7.6 Независимость от репликации

В системе поддерживается независимость от репликации, если заданное хранимое отношение или заданный фрагмент могут быть представлены несколькими различными копиями, или репликами, хранимыми на нескольких различных узлах.

15.7.7 Обработка распределенных запросов.

Вопрос оптимизации более важен для распределенной, нежели для централизованной системы. Основная причина заключается в том, что для выполнения охватывающего несколько узлов запроса существует довольно много способов перемещения данных по сети. В таком случае чрезвычайно важно найти наиболее эффективную стратегию.

15.7.8 Управление распределенными транзакциями.

Как известно, существует два главных аспекта управления транзакциями, а именно: управление восстановлением и управление параллельностью обработки. Оба этих аспекта имеют расширенную трактовку в среде распределенных систем. Чтобы разъяснить особенности этой расширенной трактовки, сначала необходимо ввести новое понятие агент. В распределенной системе отдельная транзакция может включать в себя выполнение кода на многих узлах, в частности это могут быть операции обновления, выполняемые на нескольких узлах. Поэтому говорят, что каждая транзакция содержит несколько агентов, где под агентом подразумевается процесс, который выполняется для данной транзакции на отдельном узле. Система должна знать, что два агента являются элементами одной и той же транзакции, например два агента, которые являются частями одной и той же транзакции, очевидно, не должны оказываться в состоянии взаимной блокировки.

Теперь обратимся непосредственно к управлению восстановлением. Чтобы обеспечить атомарность транзакции(принцип"все или ничего") в распределенной среде, система должна гарантировать, что все множество относящихся к данной транзакции агентов или зафиксировало свои результаты, или выполнило откат. Такого результата можно достичь с помощью протокола двухфазной фиксации транзакции.

Что касается управления параллельностью, то оно в большинстве распределенных систем базируется на механизме блокирования, точно так, как и в не распределенных системах.

15.7.9 Независимость от аппаратного обеспечения.

Парк вычислительных машин современных организаций обычно включает множество разных компьютеров– машиныIBM, машиныICL, машиныHP, персональные компьютеры, различного рода рабочие станции и т.д. Поэтому действительно существует необходимость интегрировать данные всех этих систем и предоставить пользователю "образ единой системы". Следовательно, желательно иметь возможность запускать одну и ту же СУБД на различных аппаратных платформах и, более того, добиться, чтобы различные машины участвовали в работе распределенной системы как равноправные партнеры.

15.7.10 Независимость от операционной системы.

Очевидно, что необходима не только возможность функционирования одной и той же СУБД на различных аппаратных платформах, но и возможность ее функционирования под различными операционными

системами для многих платформ– включая различные операционные системы на одном и том же оборудовании(например, чтобы версия СУБД для операционной системыMVS, версия дляUNIX и версия дляWindows NT могли совместно использоваться в одной и той же распределенной системе).

15.7.11 Независимость от сети.

Подразумевает возможность работы узлов системы в гетерогенных сетях, с использованием различного сетевого оборудования

15.7.12 Независимость от СУБД.

Эта цель подразумевает использование несколько менее точной формулировки предположения о строгой однородности. В новой форме это предположение означает, что все экземпляры СУБД на различных узлах поддерживают один и тот же интерфейс, хотя они не обязательно должны быть копиями одного и того же программного обеспечения.

15.7.13 Распространение обновления

Как указывалось выше, основной проблемой репликации данных является то, что обновление любого логического объекта должно распространяться на все хранимые копии этого объекта. Трудности возникают из-за того, что некоторый узел, содержащий данный объект, может быть недоступен(например, из-за краха системы или данного узла) именно в момент обновления.

Общая схема устранения этой проблемы(и не единственно возможная в этом случае), называемая схемой первичной копии, и состоит из следующих этапов:

1. Одна копия каждого реплицируемого объекта называется первичной копией, а все остальные– вторичными.
2. Первичные копии различных объектов находятся на различных узлах (таким образом, эта схема является распределенной).
3. Операции обновления считаются завершенными, если обновлены все первичные копии. В таком случае в некоторый момент времени узел, содержащий такую копию, несет ответственность за распространение операции обновления на вторичные копии.

15.8. Проблемы распределенных систем

Ключевая проблема распределенных систем состоит в том, что коммуникационные сети, по крайней мере сети с"большой протяженностью" или глобальные сети, пока являются медленными. Обычная глобальная сеть чаще всего имеет среднюю скорость передачи данных от5 до10 тысяч байтов в секунду. Обычный же жесткий диск имеет скорость обмена данными около5-10 миллионов байтов в секунду. (С другой стороны, некоторые локальные сети поддерживают скорость обмена данными того же порядка, что и диски.) Поэтому основная задача распределенных систем— минимизировать использование сетей, т.е. минимизировать количество и объем передаваемых сообщений. Эта задача, в свою очередь, сталкивается с проблемами в нескольких дополнительных областях. Приведем список таких областей, хотя мы и не гарантируем, что он полный.

- Обработка запросов
- Управление каталогом
- Распространение обновлений
- Управление восстановлением
- Управление параллельностью

15.9 Параллельные СУБД.

Следует четко понимать различия, существующие между распределенными и параллельными СУБД.

Параллельная СУБД – система управления базой данных, функционирующей с использованием нескольких процессоров и жестких дисков, что позволяет ей; (если это возможно) распараллеливать выполнение некоторых операций с целью повышения общей производительности обработки.

Появление параллельных СУБД было вызвано тем фактом, что системы с одним процессором оказались не способны удовлетворять растущие требования к масштабируемости, надежности и производительности обработки данных. Эффективной и экономически обоснованной альтернативой однопроцессорным СУБД стали параллельные СУБД, функционирующие одновременно на нескольких процессорах. Применение параллельных СУБД позволяет объединить несколько маломощных машин для получения такого же уровня производительности, как и в случае одной, но более мощной машины, с дополнительным выигрышем в масштабируемости и надежности системы по сравнению с однопроцессорными СУБД.

Для предоставления нескольким процессорам совместного доступа к одной и той же базе данных параллельная СУБД должна обеспечивать

управление совместным доступом к ресурсам. То, какие именно ресурсы разделяются и как это разделение реализовано на практике, непосредственно влияет на показатели производительности и масштабируемости создаваемой системы, что, в свою очередь, определяет пригодность конкретной СУБД к условиям заданной вычислительной среды и требованиям приложений. Три основных типа архитектуры параллельных СУБД представлены:

- системы с разделением памяти;
- системы с разделением дисков;
- системы без разделения вычислительных ресурсов.

Хотя параллельная система без разделения вычислительных ресурсов иногда рассматривается как распределенная СУБД, в такой системе распределение данных обусловлено лишь стремлением к повышению производительности. Более того, узлы распределенной СУБД обычно разделены географически, находятся под управлением разных администраторов и соединены между собой относительно медленными сетевыми соединениями, тогда как узлы параллельной СУБД чаще всего располагаются на одном и том же компьютере или в пределах одной и той же производственной площадки.

Системы с разделением памяти состоят из тесно связанных между собой компонентов, в число которых входит несколько процессоров, разделяющих общую системную память. Эта архитектура, называемая также архитектурой с симметричной многопроцессорной обработкой(SMP), в настоящее время получила широкое распространение и применяется для самых разных вычислительных платформ, от персональных рабочих станций, содержащих несколько параллельно работающих микропроцессоров, большихRISC-систем и вплоть до крупнейших мэйнфреймов. Эта архитектура обеспечивает быстрый доступ к данным для ограниченного набора процессоров, количество которых обычно не превосходит64. В противном случае взаимодействие по сети становится узким местом всей системы.

Системы с разделением дисков создаются из менее тесно связанных между собой компонентов. Они являются оптимальным вариантом для приложений, которые унаследовали высокую централизацию обработки и должны обеспечивать самые высокие показатели доступности и производительности. Каждый из процессоров имеет непосредственный доступ ко всем совместно используемым дисковым устройствам, но обладает собственной оперативной памятью. Как и в случае архитектуры без разделения вычислительных ресурсов, архитектура с разделением дисков исключает узкие места, связанные с совместно используемой памятью. Однако, в отличие от архитектуры без разделения вычислительных ресурсов, данная архитектура исключает упомянутые узкие места без внесения дополнительных издержек, связанных с

физическим распределением данных по отдельным устройствам. Распределенные дисковые системы иногда называют кластерами. Системы без разделения вычислительных ресурсов(эту архитектуру иногда называют архитектурой с массовой параллельной обработкой) используют схему, в которой каждый процессор, являющийся частью системы, имеет свою собственную оперативную и дисковую память. База данных распределена между всеми дисковыми устройствами, подключенным к отдельным, связанным с этой базой данных вычислительным подсистемам, в результате чего все данные прозрачно доступны пользователям каждой из этих подсистем. Такая архитектура обеспечивает более высокий уровень масштабируемости, чем системы с разделяемой памятью, и позволяет легко поддерживать большое количество процессоров. Однако оптимальной производительности удается достичь только в том случае, если требуемые данные хранятся локально. Параллельные технологии обычно используются в случае исключительно больших баз данных, размеры которых могут достигать нескольких терабайтов(1012 байт), или в системах, обеспечивающих выполнение тысяч транзакций в секунду. Подобные системы нуждаются в доступе к большому объему данных и должны обеспечивать приемлемое время реакции на запрос. Параллельные СУБД могут использовать различные вспомогательные технологии, позволяющие повысить производительность обработки сложных запросов за счет применения методов распараллеливания операций просмотра, соединения и сортировки, что позволяет нескольким процессорным узлам автоматически распределять между собой текущую нагрузку. Достаточно отметить, что все крупные разработчики СУБД в настоящее время поставляют параллельные версии созданных ими машин баз данных.

Контрольные вопросы

1. Понятие распределенной базы данных.
2. Понятие распределенная система управления базы данных.
3. Какие характеристики должна иметь любая распределенная СУБД
4. Характерные черты параллельных и распределенных СУБД
5. Преимущества распределенных базы данных
6. Однородные и разнородные распределенные СУБД
7. Функции распределенных СУБД
8. Архитектура распределенных СУБД
9. Принципы функционирования распределенной БД
10. Проблемы распределенных систем
11. Параллельные СУБД

Список литературы

1. Eric Redmond, Jim R. Wilson. A Guide to Modern Databases and the NoSQL Movement– 347 c. AQSH, 2015 г.

2. Фиайли К. SQL: Пер. с англ. – М.: ДМК Пресс. – 456 с. Москва. 2013 г.

3. Jeffrey A. Hoffer, Mary B. Prescott, and Fred R. McFadden. Modern DatabaseManagement (8th Ed.) – 557 p. Prentice-Hall, 2007.

4. For those seeking a stronger technical treatment of database systems: Elmasri, R. and S. B. Navathe: Fundamentals of Database Systems (5th Ed.) – 671 p. Addison Wesley, 2015.

5. Fundamentals of database systems sixth edition. Ramez Elmasri. Department of Computer Science and Engineering The University of Texas at Arlington. 2011.

6. Введение в Oracle 10g. Перри Джеймс, Пост Джеральд. 697 стр 2013.

7. Григорьев Ю.А., Плутенко А.Д.. Жизненный цикл проектов распределенных баз данных. Благовещенск АмГУ, 1999.

8. Диго С.М. Базы данных Проектирование и использование. издательство "Финансы и статистика" 592 стр, 2005 г.

9. Конноли Томас, Каролин Бегг. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. 3-издание - М. : Изд.дом Вильямс - 2003. - 1440 с.

ОГЛАВЛЕНИЯ

Х.Ж. Рахимбоев, У.Ф. Зарипов

БАЗЫ ДАННЫХ

Учебное пособие

Ташкент - "METODIST NASHRIYOTI" - 2024

Muharrir: Bakirov Nurmuhammad

Texnik muharrir: Tashatov Farrux
Musahhih: Muhammadiyeva Sevinch
Dizayner: Ochilova Zarnigor

Bosishga 18.05.2024.da ruxsat etildi.
Bichimi 60x90. "Times New Roman" garniturasi.
Ofset bosma usulida bosildi.
Shartli bosma tabog'i 12. Nashr bosma tabog'i 12.
Adadi 300 nusxa.

"METODIST NASHRIYOTI" MCHJ matbaa bo'limida chop etildi.
Manzil: Toshkent shahri, Shota Rustaveli 2-vagon tor ko'chasi, 1-uy.

+99893 552-11-21

Nashriyot roziligisiz chop etish ta'qiqlanadi

ISBN 978-9910-03-117-5