СЕВЕРО-КАЗАХСТАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ им. М. КОЗЫБАЕВА

Т. В. ПЯТКОВА

КУРС ЛЕКЦИЙ ПО ДИСЦИПЛИНЕ «БАЗЫ ДАННЫХ В ИНФОРМАЦИОННЫХ СИСТЕМАХ»

учебно-методическое пособие

Петропавловск
2013

УДК 004
ББК 32.937
П 99

Издается по решению учебно-методического совета Северо-Казахстанского государственного университета им. М. Козыбаева (протокол №6 от 28.02.2013 г.)

Рецензент:

директор Петропавловского филиала «Информационные технологии», к.э.н. *Л. А. Горковенко*

Пяткова Т. В.

П 99 Курс лекций по дисциплине «Базы данных в информационных системах»: учебно-методическое пособие. - Петропавловск: СКГУ им. М. Козыбаева, 2013. – 103 с.

ISBN 978-601-272-497-4

Данное учебно-методическое пособие предназначено для студентов очной и заочной формы обучения специальности «Информационные системы» 5В070300 и содержит необходимый теоретический минимум для изучения дисциплины «Базы данных в информационных системах». Курс лекций разработан в соответствии с ГОСО РК, рабочим учебным планом специальности и каталогом элективных дисциплин.

УДК 004
ББК 32.937

ISBN 978-601-272-497-4

Система менеджмента качества СКГУ им. М. Козыбаева сертифицирована на соответствие требованиям ISO 9001:2008

Содержание

Лекция № 1 «Классификация информационных систем»

Информационные системы могут классифицироваться по различным признакам. Чаще всего классификация информационных систем производится по масштабу использования, сфере применения, способу организации (архитектуре), по распределенности, по однородности и др. Сводная классификация информационных систем по различным критериям приведена на рисунке 1. Знание классификаций позволит определить предварительные характеристики информационной системы применительно к особенностям предметной области, решаемым задачам и аппаратно-программному окружению.

1.1 Классификация информационных систем по масштабу использования.

По масштабу использования информационные системы подразделяются на следующие виды:

- одиночные информационные системы;
- групповые информационные системы;
- корпоративные информационные системы.

Одиночные информационные системы реализуются, как правило, на автономном персональном компьютере, компьютерная сеть не используется. Такая система может содержать несколько простых приложений, связанных общим информационным фондом, и рассчитана на работу одного пользователя или группы пользователей, разделяющих по времени одно рабочее место. Подобные приложения создаются с помощью настольных или локальных систем управления базами данных. Среди локальных СУБД наиболее известными являются Clarion, Clipper, FoxPro, Paradox, dBase и Microsoft Access.

Групповые информационные системы ориентированы на коллективное использование информации членами рабочей группы и чаще всего строятся на базе локальной вычислительной сети. При разработке таких приложений используются серверы баз данных, называемые также SQL-серверами, для рабочих групп. Существует довольно большое

количество SQL-серверов, как коммерческих, так и свободно распространяемых. Среди них наиболее известны такие серверы баз данных, как Oracle, DB2, Microsoft SQL Server, InterBase, Sybase, Informix, MySQL.

Корпоративные информационные системы являются развитием систем для рабочих групп, они ориентированы на крупные компании и могут поддерживать территориально разнесенные узлы или сети. В основном они имеют иерархическую структуру из нескольких уровней. Для таких систем характерна архитектура клиент-сервер со специализацией серверов или же многоуровневая архитектура. При разработке таких систем могут использоваться те же серверы баз данных, что и при разработке групповых информационных систем. Однако в крупных информационных системах наибольшее распространение получили серверы Oracle, DB2 и Microsoft SQL Server.

Для групповых и корпоративных систем существенно повышаются требования к надежности функционирования и сохранности данных. Эти свойства обеспечиваются поддержкой целостности данных, ссылок и транзакций в серверах баз данных.

1.2 Классификация информационных систем по сфере применения.

По сфере применения информационные системы обычно подразделяются на четыре группы:

- информационные системы оперативной обработки транзакций (OLTP);
- информационные системы оперативной аналитической обработки данных (OLAP);
- информационно-справочные системы;
- офисные информационные системы.

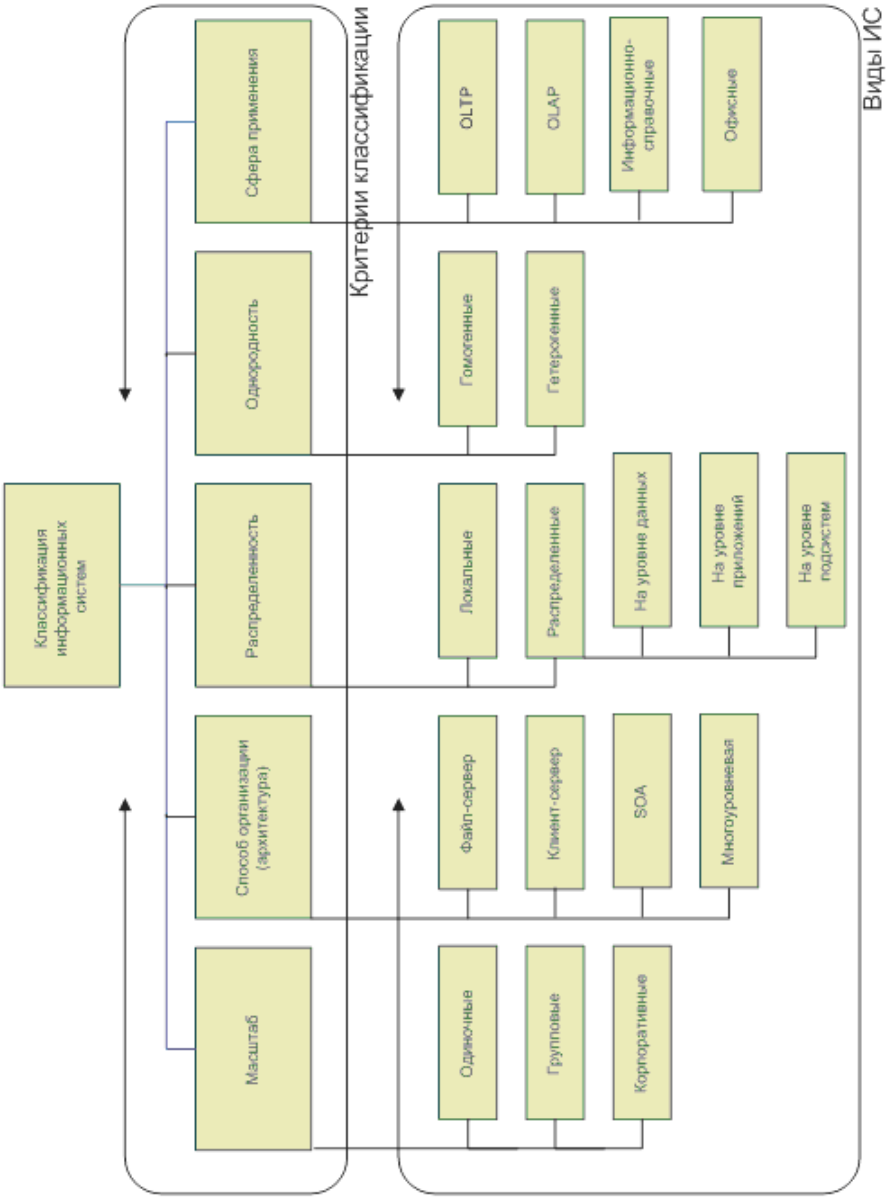

Рисунок 1. Сводная классификация информационных систем

Системы обработки транзакций, в свою очередь, по оперативности обработки данных, разделяются на пакетные информационные системы и оперативные информационные системы. В пакетных информационных системах данные об операциях накапливаются в течение некоторого периода времени и периодически обрабатываются. В оперативных информационных системах операция выполняется немедленно. В информационных системах организационного управления преобладает режим оперативной обработки транзакций (OLTP – On-Line Transaction Processing) для отражения актуального состояния предметной области и любой момент времени, а пакетная обработка занимает весьма ограниченную часть.

Типичными примерами OLTP-приложений являются системы производственные системы, складского учета, системы заказов билетов, банковские системы, выполняющие операции по переводу денег, и т.п. Основная функция подобных систем заключается в выполнении большого количества коротких транзакций. Сами транзакции выглядят относительно просто, например, "снять сумму денег со счета А, добавить эту сумму на счет В".

Для OLTP-систем, в общем случае, характерно следующее:

1. большое количество несложных транзакций, как правило, связанных с добавлением и модификацией данных;
2. к системе может быть подключено большое количество одновременно работающих пользователей (в крупных системах до несколько тысяч), поэтому транзакции должны выполняться одновременно;
3. при возникновении ошибки, транзакция должна целиком откатиться и вернуть систему к состоянию, которое было до начала транзакции – не должно быть, например, ситуации, когда деньги сняты со счета А, но не поступили на счет В.

Практически все запросы к базе данных в OLTP-приложениях состоят из команд вставки, обновления, удаления. Запросы на выборку в основном предназначены для

предоставления пользователям возможности выбора из различных справочников. Большая часть запросов, как правило, известна заранее еще на этапе проектирования системы. Таким образом, критическим для OLTP-приложений является скорость и надежность выполнения коротких операций обновления данных.

Другим типом информационных систем являются системы оперативной аналитической обработки данных (OLAP – On-Line Analitical Processing). Это обобщенный термин, характеризующий принципы построения систем поддержки принятия решений (Decision Support System – DSS), хранилищ данных (Data Warehouse), систем интеллектуального анализа данных (Data Mining). Такие системы предназначены для нахождения зависимостей между данными. Например, можно попытаться определить, как связан объем продаж товаров с характеристиками потенциальных покупателей, для проведения анализа "что если…". OLAP-приложения оперируют с большими массивами данных, уже накопленными в OLTP-приложениях, взятыми их электронных таблиц или из других источников данных. Такие системы характеризуются следующими признаками:

1. добавление в систему новых данных происходит относительно редко крупными блоками (например, раз в квартал загружаются данные по итогам квартальных продаж из OLTP-приложения);
2. данные, добавленные в систему, удаляются редко или не удаляются никогда;
3. перед загрузкой данные проходят различные процедуры "очистки", связанные с тем, что в одну систему могут поступать данные из многих источников, имеющих различные форматы представления для одних и тех же понятий, данные могут быть некорректны, либо ошибочны;
4. запросы к системе являются нерегламентированными и, как правило, достаточно сложными, запросы могут формулироваться аналитиком в процессе работы в системе;

5. скорость выполнения запросов важна, но не критична.

Данные OLAP-приложений обычно представлены в виде одного или нескольких гиперкубов, измерения которого представляют собой справочные данные, а в ячейках самого гиперкуба хранятся собственно данные. Например, можно построить гиперкуб, измерениями которого являются: время (в кварталах, годах), тип товара и отделения компании, а в ячейках хранятся объемы продаж. Такой гиперкуб будет содержать данных о продажах различных типов товаров по кварталам и подразделениям. Основываясь на этих данных, можно отвечать на вопросы вроде "у какого подразделения самые лучшие объемы продаж в текущем году?", или "каковы тенденции продаж отделений Юго-Западного региона в текущем году по сравнению с предыдущим годом?".

OLAP-приложения являются составной частью управленческих информационных систем, предназначенных для выработки эффективных управленческих решений.

Обширный класс информационно-справочных систем основан на гипертекстовых документах и мультимедиа. Наибольшее развитие такие информационные системы получили в сети Интернет.

Класс офисных информационных систем нацелен на перевод бумажных документов в электронный вид, автоматизацию делопроизводства и управление документооборотом.

Следует отметить, что приводимая классификация по сфере применения в достаточной степени условна. Крупные информационные системы очень часто обладают признаками всех перечисленных выше классов. Кроме того, корпоративные информационные системы масштаба предприятия обычно состоят из ряда подсистем, относящихся к различным сферам применения.

1.3 Классификация информационных систем по способу организации (архитектуре).

По способу организации информационные системы подразделяются на следующие классы:

1. информационные системы с централизованной архитектурой;
2. информационные системы с архитектурой клиент-сервер;
3. информационные системы с сервис-ориентированной архитектурой (SOA);
4. информационные системы с многоуровневой архитектурой.

В любой информационной системе можно выделить функциональные компоненты, которые помогают понять ограничения различных архитектур информационных систем. На верхнем уровне абстрагирования можно выделить компоненты, представленные в таблице 1.

Презентационная логика как часть приложения определяется тем, что пользователь видит на своем экране, что приложение работает. Сюда относятся все интерфейсные экранные формы, которые пользователь видит или заполняет в ходе работы приложения, а также все то, что выводится пользователю на экран в качестве результатов решения некоторых промежуточных задач либо как справочная информация. Следовательно, основными задачами презентационной логики являются:

1. формирование экранных изображений;
2. чтение и запись в экранные формы информации;
3. управление экраном;
4. обработка движений мыши и нажатие клавиш клавиатуры.

Бизнес-логика, или логика собственно приложений, – это часть кода приложения, которая определяет собственно алгоритмы решения конкретных его задач. Обычно этот код записывается с использованием различных языков программирования, таких как C, C++, Visual Basic и др.

Таблица 1 – Функциональные компоненты информационных систем

№ п/п	Обозначение	Наименование	Характеристики
1	PS	Presentation Services (средства представления)	Обеспечиваются устройствами, принимающими ввод от пользователя и отображающими то, что сообщает ему компонент логики представления PL, с использованием соответствующей программной поддержки
2	PL	Presentation Logic (логика представления)	Управляет взаимодействием между пользователем и ЭВМ. Обрабатывает действия пользователя при выборе команды в меню, нажатии кнопки или выборе элемента из списка
3	BL	Business or Application Logic (логика приложений)	Набор правил для принятия решений, вычислений и операций, которые должно выполнить приложение

4	DL	Data Logic (логика обработки данных)	Операции с базой данных (например, SQL-операторы), которые нужно выполнить для реализации прикладной логики
5	DS	Data Services (операции с базой данных)	Действия СУБД, вызываемые для выполнения логики управления данными, такие как, манипулирование данными, определение данных, фиксация или откат транзакций и т. п.
6	FS	File Services (файловые операции)	Дисковые операции чтения и записи данных для СУБД и других компонентов. Обычно являются функциями операционной системы (ОС)

Логика обработки данных – это часть кода приложения, которая непосредственно связана с обработкой данных внутри него. Данными управляет собственно СУБД, а для обеспечения доступа к ним используется язык манипулирования данными. Для реляционных СУБД – это язык SQL.

Логика управления информационными ресурсами или управления базой данных реализуются собственно СУБД, которая обеспечивает хранение и управление данными в базах

данных. В идеале функции СУБД должны быть скрыты от бизнес-логики приложения, однако при рассмотрении архитектуры приложения мы выделим их в отдельную его часть.

В централизованной архитектуре (Host-Based Processing) указанные части приложения располагаются в единой среде и комбинируются внутри одной исполняемой программы. На основе централизованной архитектуры могут создаваться только одиночные информационные системы, устанавливаемые и функционирующие локально на персональных компьютерах конечных пользователей.

Для такого способа организации не требуется оддержки сети и все сводится к автономной работе. Работа построена следующим образом:

База данных в виде набора файлов находится на жестком диске компьютера;

На том же компьютере установлены СУБД и приложение для работы с БД;

Пользователь запускает приложение. Используя предоставляемый приложением пользовательский интерфейс, он инициирует обращение к БД на выборку и/или обновление информации;

Все обращения к БД идут через СУБД, которая инкапсулирует внутри себя все сведения о физической структуре БД;

СУБД инициирует обращения к данным, обеспечивая выполнение запросов пользователя (осуществляя необходимые операции над данными);

Результат СУБД возвращает в приложение;

Приложение, используя пользовательский интерфейс, отображает результат выполнения запросов.

Подобная архитектура использовалась в первых версиях СУБД DB2, Oracle, Ingres.

Многопользовательская технология работы обеспечивалась либо режимом мультипрограммирования, при котором одновременно могли работать процессор и внешние устройства –например, пока в прикладной программе одного

пользователя шло считывание данных из внешней памяти, программа другого пользователя обрабатывалась процессором, либо режимом разделения времени, при котором пользователям по очереди выделялись кванты времени на выполнении их программ. Такая технология была распространена в период «господства» больших ЭВМ (IBM-370, ЕС-1045, ЕС-1060). Основным недостатком этой модели является резкое снижение производительности при увеличении числа пользователей.

В децентрализованной архитектуре указанные части приложения могут быть по-разному распределены между серверным и клиентским процессами. На основе децентрализованной архитектуры создаются групповые и корпоративные информационные системы.

Вычислительная модель клиент-сервер исходно связана с появлением открытых систем в 1990-х гг. Термин клиент-сервер применялся к архитектуре программного обеспечения, состоящего из двух процессов обработки информации: клиентского и серверного. Клиентский процесс запрашивал некоторые услуги, а серверный – обеспечивал их выполнение. При этом предполагалось, что один серверный процесс может обслужить множество клиентских процессов. Учитывая, что аппаратная реализация этой модели управления базами данных связана с созданием локальных вычислительных сетей предприятия, такую организацию процесса обработки информации называют архитектурой клиент-сервер.

В зависимости от характера распределений задач можно выделить следующие модели архитектуры клиент-сервер:

1. распределенное представление (Distribution Presentation);
2. удаленное представление (Remote Presentation);
3. распределенная бизнес-логика (Remote Business Logic);
4. удаленное управление данными (Remote Data Management);
5. распределенное управление данными (Distributed DataManagement).

Эта условная классификации показывает, как могут быть распределены отдельные задачи между серверным и клиентскими процессами. В данной классификации отсутствует реализация удаленной бизнес-логики, так как считается, что она не может быть удалена полностью, а может быть лишь распределена между разными процессами, которые могут взаимодействовать друг с другом.

1.4 Классификация информационных систем по распределнности.

По распределенности различают локальные, предназначенные для работы на автономном компьютере и распределенные информационные системы (РИС), состоящие из нескольких подсистем (информационных узлов), располагающих программно-аппаратными средствами реализации информационных технологий и множества средств, обеспечивающих соединение и взаимодействие этих подсистем.

1.5 Классификация информационных систем по однородности.

По однородности ИС можно разделить на гомогенные (homogencous) и гетерогенные (heterogencous). Гетерогеность и гомогенность можно рассматривать в нескольких аспектах, касающихся использования языков программирования, операционных систем, представления данных, взаимодействия частей системы и др.

Лекция 2 «История возникновения и развития баз данных»

2.1 Данные, предметная область, база данных

Прежде чем приступить к изучению принципов организации баз данных рассмотрим базовые понятия курса, такие как «данные, предметная область, база данных».

С точки зрения пользователей автоматизированных информационных систем данные являются информацией, фиксированной в определенной форме, пригодной для последующей обработки и хранения.

Как правило, набор данных относится к определенной области знаний объектов, событий. Такую область принято называть предметной областью для данного набора данных.

Для того, чтобы данные можно было использовать для решения практических задач их нужно научиться хранить. Понятие «база данных» с этой точки зрения можно определить как средство (инструмент) для хранения и последующего использования данных.

Примерами баз данных могут быть самые разные средства для хранения данных.

Каждый человек на протяжении своей жизни пользуется разнообразными базами данных, удовлетворяя потребности в конкретной информации. Век электроники и компьютерной техники кардинально изменил возможности хранения и последующего использования данных. Появились такие средства для хранения данных как мобильный телефон, компьютер, магнитофон и целый ряд других устройств. Однако важнейшее значение для организации баз данных имеют компьютеры. Само понятие базы данных появилось благодаря компьютерам.

2.2 Появление компьютерных баз данных

Базы данных как область использования вычислительной техники возникла несколько позже первой, связанной с вычислениями. Это связано с тем, основной функцией баз данных является хранение больших объемов информации, а на заре вычислительной техники возможности компьютеров по хранению информации были очень ограниченными. Говорить о надежном и долговременном хранении информации можно только при наличии запоминающих устройств, сохраняющих информацию после выключения электрического питания. Оперативная (основная) память компьютеров этим свойством обычно не обладает. В первых компьютерах использовались два вида устройств внешней памяти — магнитные ленты и барабаны. Емкость магнитных лент была достаточно велика, но по своей физической природе они обеспечивали последовательный доступ к данным. Магнитные же барабаны

(они ближе всего к современным магнитным дискам с фиксированными головками) давали возможность произвольного доступа к данным, но имели ограниченный объем хранимой информации.

Эти ограничения не являлись слишком существенными для чисто численных расчетов. Даже если программа должна обработать (или произвести) большой объем информации, при программировании можно продумать расположение этой информации во внешней памяти (например, на последовательной магнитной ленте), обеспечивающее эффективное выполнение этой программы. Однако в базах данных требуется сравнительно быстрая реакции системы на запросы пользователей. И в этом случае наличие сравнительно медленных устройств хранения данных, к которым относятся магнитные ленты и барабаны, было недостаточным.

Можно предположить, что именно требования нечисловых приложений вызвали появление съемных магнитных дисков с подвижными головками, что явилось революцией в истории вычислительной техники. Эти устройства внешней памяти обладали существенно большей емкостью, чем магнитные барабаны, обеспечивали удовлетворительную скорость доступа к данным в режиме произвольной выборки, а возможность смены дискового пакета на устройстве позволяла иметь практически неограниченный архив данных.

С появлением магнитных дисков началась история систем управления данными во внешней памяти. До этого каждая прикладная программа, которой требовалось хранить данные во внешней памяти, сама определяла расположение каждой порции данных на магнитной ленте или барабане и выполняла обмены между оперативной памятью и устройствами внешней памяти с помощью программно-аппаратных средств низкого уровня (машинных команд или вызовов соответствующих программ операционной системы). Такой режим работы не позволяет или очень затрудняет поддержание на одном внешнем носителе нескольких архивов долговременно хранимой информации. Кроме того, каждой прикладной

программе приходилось решать проблемы именования частей данных и структуризации данных во внешней памяти.

Важным шагом в развитии баз данных явился переход к использованию централизованных систем управления файлами. С точки зрения прикладной программы, файл — это именованная область внешней памяти, в которую можно записывать и из которой можно считывать данные. Правила именования файлов, способ доступа к данным, хранящимся в файле, и структура этих данных зависят от конкретной системы управления файлами и, возможно, от типа файла. Система управления файлами берет на себя распределение внешней памяти, отображение имен файлов в соответствующие адреса во внешней памяти и обеспечение доступа к данным.

Пользователи видят файл как линейную последовательность записей и могут выполнить над ним ряд стандартных операций:

1. создать файл (требуемого типа и размера);
2. открыть ранее созданный файл;
3. прочитать из файла некоторую запись (текущую, следующую, предыдущую, первую, последнюю);
4. записать в файл на место текущей записи новую, добавить новую запись в конец файла.

В разных файловых системах эти операции могли несколько отличаться, но общий смысл их был именно таким. Главное, что следует отметить, это то, что структура записи файла была известна только программе, которая с ним работала, система управления файлами не знала ее. И поэтому для того, чтобы извлечь некоторую информацию из файла, необходимо было точно знать структуру записи файла с точностью до бита. Каждая программа, работающая с файлом, должна была иметь у себя внутри структуру данных, соответствующую структуре этого файла. Поэтому при изменении структуры файла требовалось изменять структуру программы, а это требовало новой компиляции, то есть процесса перевода программы в исполняемые машинные коды. Такая ситуации характеризовалась как зависимость программ от данных.

Для баз данных характерным является наличие большого числа различных пользователей (программ), каждый из которых имеет свои специфические алгоритмы обработки информации, хранящейся в одних и тех же файлах. Изменение структуры файла, которое было необходимо для одной программы, требовало исправления и перекомпиляции и дополнительной отладки всех остальных программ, работающих с этим же файлом. Это было первым существенным недостатком файловых систем, который явился толчком к созданию новых систем хранения и управления информацией.

Далее, поскольку файловые системы являются общим хранилищем файлов, принадлежащих, вообще говоря, разным пользователям, системы управления файлами должны обеспечивать авторизацию доступа к файлам. В общем виде подход состоит в том, что по отношению к каждому зарегистрированному пользователю данной вычислительной системы для каждого существующего файла указываются действия, которые разрешены или запрещены данному пользователю. В большинстве современных систем управления файлами применяется подход к защите файлов, впервые реализованный в ОС UNIX. В этой ОС каждому зарегистрированному пользователю соответствует пара целочисленных идентификаторов: идентификатор группы, к которой относится этот пользователь, и его собственный идентификатор в группе. При каждом файле хранится полный идентификатор пользователя, который создал этот файл, и фиксируется, какие действия с файлом может производить его создатель, какие действия с файлом доступны для других пользователей той же группы и что могут делать с файлом пользователи других групп. Администрирование режимом доступа к файлу в основном выполняется его создателем-владельцем. Для множества файлов, отражающих информационную модель одной предметной области, такой децентрализованный принцип управления доступом вызывал дополнительные трудности. И отсутствие централизованных методов управления доступом к информации послужило еще

одной причиной разработки специальных программ для работы с БД.

Следующей причиной стала необходимость обеспечения эффективной параллельной работы многих пользователей с одними и теми же файлами. В общем случае системы управления файлами обеспечивали режим многопользовательского доступа. Если операционная система поддерживает многопользовательский режим, вполне реальна ситуация, когда два или более пользователя одновременно пытаются работать с одним и тем же файлом. Если все пользователи собираются только читать файл, ничего страшного не произойдет. Но если хотя бы один из них будет изменять файл, для корректной работы этих пользователей требуется взаимная синхронизация их действий по отношению к файлу.

При подобном способе организации одновременная работа нескольких пользователей, связанная с модификацией данных в файле, либо вообще не реализовывалась, либо была очень замедлена.

Эти недостатки послужили тем толчком, который заставил разработчиков информационных систем предложить новый подход к управлению информацией. Этот подход был реализован в рамках новых программных систем, названных впоследствии Системами Управления Базами Данных (СУБД), а сами хранилища информации, которые работали под управлением данных систем, назывались базами данных.

2.3 Базы данных в Excel

Одним из простых и удобных способов хранения и обработки данных могут служить электронные таблицы, например программа Microsoft Excel. Эта программа обладает многими возможностями, присущими современным программам для работы с базами данных. Так, например, Excel имеет средства для обработки данных и осуществления их фильтрации. Однако по своим возможностям Excel уступает специализированным программам, предназначенным для работы с базами, таким как Microsoft Access, Oracle, SQL. В частности, Excel не может работать с очень большими базами данных. Это

связано с тем, что программа Excel загружает в память всю базу данных, независимо от ее объема. Специализированные же программы загружают только те данные, которые необходимы.

Внутренней базой данных в Excel являются списки. Списки практически ничем не отличаются от того, с чем пользователь обычно имеет дело, работая в Excel: здесь информация также сохраняется в строках и столбцах. Согласно терминологии, принятой в базах данных, понятие поле эквивалентно принятому в Excel понятию столбец. Метки, описывающие поля, называются именами полей. Понятие запись эквивалентно принятому в Excel понятию строка.

При осуществлении фильтрации списка согласно установленному критерию или правилам часть его данных оказываются "спрятанными" (на экране их не видно). Предположим, что у нас имеется список всех клиентов некой фирмы из России и стран СНГ. Мы можем отфильтровать наш список так, что в нем останутся клиенты только из Украины. Или же этот список можно отфильтровать так, что в нем останутся только те клиенты, заказы которых превышают некоторую заранее установленную сумму. Для каждого конкретного списка имеется много возможных путей его фильтрации. Фильтрация данных в списках Excel осуществляется с помощью двух инструментов: Автофильтр и Расширенный фильтр.

Для организации данных можно воспользоваться имеющимися в Excel инструментами сортировки. В зависимости от типа данных, Excel может задать ряд вопросов с тем, чтобы определить, по какому принципу необходимо сортировать эти данные. В отличии от фильтрации данных, операция сортировки не приводит к скрытию данных. Можно видеть те же данные, но в другом порядке.

Предположим, что на рабочем листе Excel список товаров. Можно воспользоваться инструментом Сортировка для того, чтобы разместить все эти товары в алфавитном порядке, начиная с буквы А. Для выполнения сортировки списка товаров в алфавитном порядке, начиная с буквы А, надо выделить

колонку, которую нужно отсортировать и щелкнуть на стандартной панели инструментов кнопку Сортировка по возрастанию.

Для численных значений щелчок на этой кнопке приведет к тому, что данные расположатся по возрастанию численных значений: начиная с самого меньшего и кончая самым большим числом.

Если нужно расположить список товаров в обратном алфавитном порядке, то следует воспользоваться кнопкой Сортировка по убыванию. Использование этой кнопки для численных значений приведет к тому, что вверху столбца расположится самое большое численное значение, а внизу самое маленькое.

Отсортировать данные можно также и в том случае, если они расположены не в столбце, а в строке. Для этого необходимо просто выделить эти данные и щелкнуть по кнопке Сортировка по убыванию или Сортировка по возрастанию. При сортировке данных программа Excel использует действительное значение, а не ту величину, которая показана на экране. Эти численные данные могут содержать до пятнадцати знаков после запятой, и именно эти значения берет программа Excel при создании упорядоченного списка.

В Excel есть возможность для создания своих списков автозаполнения и сохранения их для дальнейшего использования. Например, можно создать список государств и затем сохранить его как список для автозаполнения. Теперь, после того как введены два первых государства, инструмент Автозаполнение заполнит весь список. Можно также создать список месяцев в году или список дней недели

2.4 Системы управления базами данных (СУБД)

Для хранения и обработки данных можно использовать различные компьютерные программные средства, например рассмотренную выше программу Excel. Однако по мере усложнения задач хранения и обработки данных, увеличения объемов хранимой и обрабатываемой информации стали разрабатываться специальные программные средства. Вначале в

качестве программных средств для разработки баз данных использовались программы общего назначения, такие как паскаль, бейсик, фортран и другие. Но постепенно, по мере роста востребованности такого рода программного обеспечения стали создаваться специальные инструментальны для разработки и работы с данными. Такие специализированные программы получили название системы управления базами данных (СУБД).

История развития СУБД насчитывает более 40 лет. В 1968 году была введена в эксплуатацию первая промышленная СУБД система IMS фирмы IBM. В 1975 году появился первый стандарт ассоциации по языкам систем обработки данных — Conference of Data System Languages (CODASYL), который определил ряд фундаментальных понятий в теории систем баз данных, которые и до сих пор являются основополагающими для сетевой модели данных. В дальнейшее развитие теории баз данных большой вклад был сделан американским математиком Э. Ф. Коддом, который является создателем реляционной модели данных.

Первый этап развития СУБД связан с организацией баз данных на больших машинах типа IBM 360/370, ЕС-ЭВМ и мини-ЭВМ типа PDP11 (фирмы Digital Equipment Corporation — DEC), разных моделях HP (фирмы Hewlett Packard). Базы данных хранились во внешней памяти центральной ЭВМ, пользователями этих баз данных были задачи, запускаемые в основном в пакетном режиме. Интерактивный режим доступа обеспечивался с помощью консольных терминалов, которые не обладали собственными вычислительными ресурсами (процессором, внешней памятью) и служили только устройствами ввода-вывода для центральной ЭВМ. Программы доступа к БД писались на различных языках и запускались как обычные числовые программы. Мощные операционные системы обеспечивали возможность условно параллельного выполнения всего множества задач. Эти системы можно было отнести к системам распределенного доступа, потому что база данных была централизованной, хранилась на устройствах внешней

памяти одной центральной ЭВМ, а доступ к ней поддерживался от многих пользователей-задач.

Появление персональных компьютеров произвело настоящую революцию в создании систем управления данными. Постоянное снижение цен на персональные компьютеры сделало их доступными не только для организаций и фирм, но и для отдельных пользователей. Компьютеры стали инструментом для ведения документации и собственных учетных функций. Это все сыграло как положительную, так и отрицательную роль в области развития баз данных. Кажущаяся простота и доступность персональных компьютеров и их программного обеспечения породила множество дилетантов. Эти разработчики, считая себя знатоками, стали проектировать недолговечные базы данных, которые не учитывали многих особенностей объектов реального мира. Много было создано систем-однодневок, которые не отвечали законам развития и взаимосвязи реальных объектов. Однако доступность персональных компьютеров заставила пользователей из многих областей знаний, которые ранее не применяли вычислительную технику в своей деятельности, обратиться к ним. И спрос на развитые удобные программы обработки данных заставлял поставщиков программного обеспечения поставлять все новые системы, которые принято называть настольными (desktop) СУБД. Значительная конкуренция среди поставщиков заставляла совершенствовать эти системы, предлагая новые возможности, улучшая интерфейс и быстродействие систем, снижая их стоимость. Наличие на рынке большого числа СУБД, выполняющих сходные функции, потребовало разработки методов экспорта-импорта данных для этих систем и открытия форматов хранения данных.

На сегодняшний день на рынке программного обеспечения имеются десятки разнообразных СУБД. Все крупные компании, разрабатывающие программное обеспечение, как правило, имеют в своем активе одну или несколько СУБД. В качестве примера можно назвать такие распространенные системы как Paradox, dBase, Clipper, Access,

Clarion, Oracle, FoxPro, SQL и целый ряд аналогичных продуктов.

Современные СУБД как правило многопользовательские и предназначены для работы в локальной сети. По технологии обработки данных БД подразделяют на централизованные и распределенные.

Централизованная БД хранится в памяти одной вычислительной системы. Такой способ часто применяют в локальных сетях.

Распределенная БД состоит из нескольких, возможно пересекающихся или дублирующих друг друга частей, хранимых на разных ЭВМ. Работа с такой БД осуществляется с помощью системы управления распределенной БД.

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

Системы централизованных БД с сетевым доступом предполагают различные архитектуры подобных систем:

1. легко и удобно работать с большими объемами информации
2. осуществлять быстрый поиск и сортировку данных
3. представлять данные в различных видах
4. вносить изменения в данные, добавлять, удалять записи, менять структуру базы
5. обмениваться информацией с другими базами
6. выводить на принтер или другие носители информацию из БД
7. оформление почтовой корреспонденции, получение готовых форм различной отчетной документации

2.5 Модели представления данных в СУБД

Прежде всего попробуем разобраться, что такое модель и какие модели используются в системах управления базами данных?

Модель - это некоторое упрощенное подобие реального объекта, которое в определенных условиях может его заменить. Модели по своей природе могут быть самые разные, например физические модели, математические, информационные.

Примером физической модели может служить модель автомобиля или здания. Математические формулы, описывающие поведение некоторых реальных объектов являются математическими моделями этих объектов. Нас же прежде всего будут интересовать информационные модели.

Информационная модель - это информация (знания, сведения) о реальном объекте, процессе, явлении. Или другое определение: Информационная модель - это набор величин, содержащих всю необходимую информацию об исследуемых объектах (процессах, явлениях).

В любой базе данных данные должны быть определенным образом структурированы. Т.е. должна существовать информационная модель, определяющая порядок организации данных в базе. На сегодня наибольшее распространение получили три модели организации данных: иерархическая, сетевая, реляционная.

Сначала стали использовать иерархические модели. Такая модель может быть представлена направленным графом. Иерархическая модель данных строится по принципу иерархии типов объектов, то есть один тип объектов является главным, а остальные, находящиеся на низших уровнях иерархии подчинёнными.

Между главным и подчинёнными объектами устанавливается взаимосвязь «один ко многим». Для каждого подчинённого типа объекта может быть только один исходный. Наивысший в иерархии узел называется корневым. Иерархическая модель позволяет строить базы данных с иерархической древовидной структурой. Например, на рисунке 1.4 объект "Организация" - предок для объектов "Отделы" и "Филиалы".

Основное достоинство иерархической модели - простота описания иерархических структур реального мира.

В сетевой модели данных понятие главного и подчинённых объектов несколько расширены. Любой объект может быть и главным и подчинённым. В сетевой модели главный объект обозначается термином «член набора». Один и

тот же объект может одновременно выступать и в роли владельца, и роли члена набора. Это означает, что любой объект может участвовать в любом числе взаимосвязей.

При разработке сетевых моделей было выдумано множество "маленьких хитростей", позволяющих увеличить производительность СУБД, но существенно усложнивших последние. Прикладной программист должен знать массу терминов, изучить несколько внутренних языков СУБД, детально представлять логическую структуру базы данных для осуществления навигации среди различных экземпляров, наборов, записей и т.п. Один из разработчиков операционной системы UNIX сказал "Сетевая база – это самый верный способ потерять данные".

В настоящее время наибольшее распространение получили реляционные базы данных. СУБД реляционного типа являются наиболее распространенными на всех классах ЭВМ, а на ПК занимают доминирующее положение.

Реляционной называется СУБД, в которой средства управления БД поддерживают реляционную модель данных. Концепция реляционной модели была предложена в 1970 Г. Е. Коддом и имеет большое значение в деле организации работы с БД.

В реляционной модели данных объекты и взаимосвязи между ними представляются с помощью таблиц. Каждая из таблиц представляет собой один объект и состоит из строк и столбцов. В реляционной базе данных каждая таблица должна иметь первичный ключ (ключевой элемент) – поле или комбинацию полей, которые единственным образом идентифицируют каждую строку в таблице. За счёт ключевых полей имеется возможность установить связи между таблицами. В целом вся БД представляет собой набор взаимосвязанных между собой таблиц.

С созданием реляционных баз данных был начат новый этап в эволюции СУБД. Простота и гибкость модели привлекли к ней внимание разработчиков и снискали ей множество сторонников. Несмотря на некоторые недостатки, реляционная

модель данных стала доминирующей, а реляционные СУБД стали промышленным стандартом "де-факто".

Рассмотрим последний вопрос этого раздела, а именно вопрос о пользователях баз данных и их основным функциям по отношению к базам данных..

Пользователями БД являются четыре основные категории потребителей ее информации и поставщиков информации для нее:

- конечные пользователи;
- программисты и системные аналитики;
- персонал поддержки БД в актуальном состоянии;
- администратор БД.

Конечным пользователям для обеспечения доступа к информации БД предоставляется графический интерфейс, как правило, в виде системы окон с функциональными меню, позволяющими легко получать необходимую информацию на экран и/или принтер в виде удобно оформленных отчетов.

Программисты и системные аналитики используют БД совершенно в ином качестве, обеспечивая разработку новых БД-приложений, поддерживая и модифицируя (при необходимости) уже существующие. Для данной группы пользователей БД требуются средства, обеспечивающие указанные функции (создание, отладка, редактирование и т.д.).

Пользователи третьей категории нуждаются в интерфейсе, как правило, графическом для обеспечения задач поддержания БД в актуальном состоянии. Эти пользователи состоят в штатах подразделений функциональных и/или обработки информации, обеспечивающих прикладную область, и отвечают за актуальное состояние соответствующей ей БД (контроль текущего состояния, удаление устаревшей информации, добавление новой и т.д.).

Особую и ответственную роль выполняет администратор, отвечающий как за актуальность находящейся в БД информации, так и за корректность функционирования и использования БД.

Лекция 3 «Введение в реляционную модель данных»

Принято считать, что реляционный подход к организации баз данных был заложен в конце 1960-х гг. Эдгаром Коддом [2.1]. В последние десятилетия этот подход является наиболее распространенным (с оговоркой, что в называемых в обиходе реляционными системах баз данных, основанных на языке SQL, в действительности нарушаются некоторые важные принципы классического реляционного подхода). Достоинствами реляционного подхода принято считать следующие свойства: реляционный подход основывается на небольшом числе интуитивно понятных абстракций, на основе которых возможно простое моделирование наиболее распространенных предметных областей; эти абстракции могут быть точно и формально определены; теоретическим базисом реляционного подхода к организации баз данных служит простой и мощный математический аппарат теории множеств и математической логики; реляционный подход обеспечивает возможность ненавигационного манипулирования данными без необходимости знания конкретной физической организации баз данных во внешней памяти. Компьютерный мир далеко не сразу признал реляционные системы. В 70-е года прошлого века, когда уже были получены почти все основные теоретические результаты и даже существовали первые прототипы реляционных СУБД, многие авторитетные специалисты отрицали возможность добиться эффективной реализации таких систем. Однако преимущества реляционного подхода и развитие методов и алгоритмов организации и управления реляционными базами данных привели к тому, что к концу 80-х годов реляционные системы заняли на мировом рынке СУБД доминирующее положение. В этой лекции на сравнительно неформальном уровне вводятся основные понятия реляционных баз данных, а также определяется сущность реляционной модели данных. Основной целью лекции является демонстрация простоты и возможности интуитивной интерпретации этих понятий. В следующих лекциях будут приводиться более

формальные определения, на которых основана теория реляционных баз данных.

Основные понятия реляционных баз данных

Выделим следующие основные понятия реляционных баз данных: тип данных, домен, атрибут, кортеж, отношение, первичный ключ.

Для начала покажем смысл этих понятий на примере отношения СЛУЖАЩИЕ, содержащего информацию о служащих некоторого предприятия – рисунок 2.

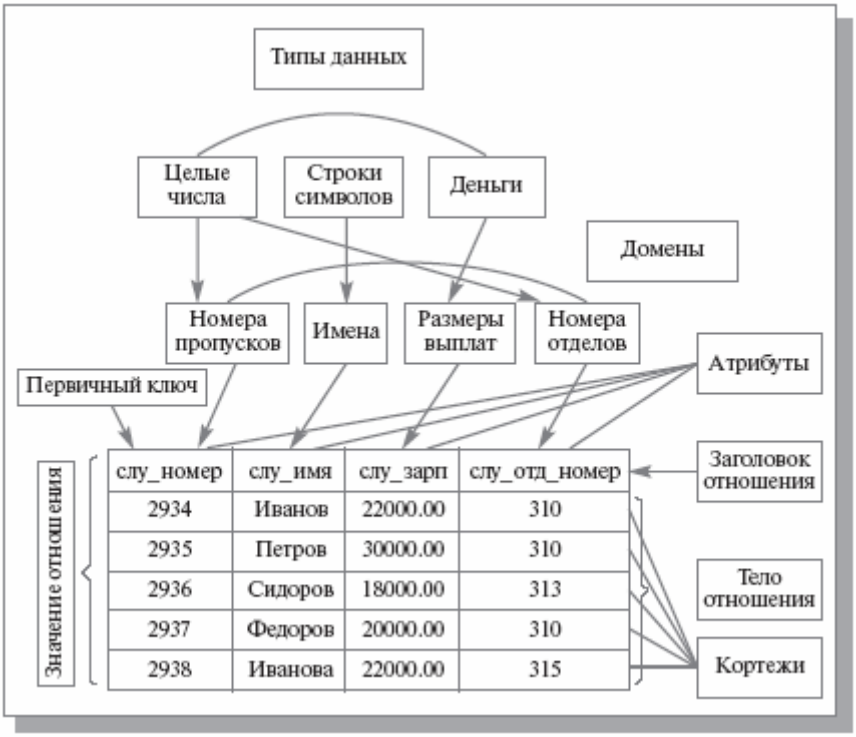

Рисунок 2. Соотношение основных понятий реляционного подхода.

Тип данных. Значения данных, хранимые в реляционной базе данных, являются типизированными, т. е. известен тип

каждого хранимого значения. Понятие типа данных в реляционной модели данных полностью соответствует понятию типа данных в языках программирования. Напомним, что традиционное (нестрогое) определение типа данных состоит из трех основных компонентов: определение множества значений данного типа; определение набора операций, применимых к значениям типа; определение способа внешнего представления значений типа (литералов).

Обычно в современных реляционных базах данных допускается хранение символьных, числовых данных (точных и приблизительных), специализированных числовых данных (таких, как «деньги»), а также специальных «темпоральных» данных (дата, время, временной интервал). Кроме того, в реляционных системах поддерживается возможность определения пользователями собственных типов данных.

В примере мы имеем дело с данными трех типов: строки символов, целые числа и «деньги».

Домен

Понятие домена более специфично для баз данных, хотя и имеются аналогии с подтипами в некоторых языках программирования (более того, в своем «Третьем манифесте» [1.5], [2.8], [3.3] Кристофер Дейт и Хью Дарвен вообще ликвидируют различие между доменом и типом данных). В общем виде домен определяется путем задания некоторого базового типа данных, к которому относятся элементы домена, и произвольного логического выражения, применяемого к элементу этого типа данных (ограничения домена). Элемент данных является элементом домена в том и только в том случае, если вычисление этого логического выражения дает результат истина (для логических значений мы будем попеременно использовать обозначения истина и ложь или true и false). С каждым доменом связывается имя, уникальное среди имен всех доменов соответствующей базы данных.

Наиболее правильной интуитивной трактовкой понятия домена является его восприятие как допустимого потенциального, ограниченного подмножества значений

данного типа. Например, домен ИМЕНА в нашем примере определен на базовом типе символьных строк, но в число его значений могут входить только те строки, которые могут представлять имена (в частности, для возможности представления русских имен такие строки не могут начинаться с мягкого или твердого знака и не могут быть длиннее, например, 20 символов). Если некоторый атрибут отношения определяется на некотором домене (как, например, на рис. 3.1 атрибут СЛУ_ИМЯ определяется на домене ИМЕНА), то в дальнейшем ограничение домена играет роль ограничения целостности, накладываемого на значения этого атрибута.

Следует отметить также семантическую нагрузку понятия домена: данные считаются сравнимыми только в том случае, когда они относятся к одному домену. В нашем примере значения доменов НОМЕРА ПРОПУСКОВ и НОМЕРА ОТДЕЛОВ относятся к типу целых чисел, но не являются сравнимыми (допускать их сравнение было бы бессмысленно).

Заголовок отношения, кортеж, тело отношения, значение отношения, переменная отношения.

Понятие отношения является наиболее фундаментальным в реляционном подходе к организации баз данных, поскольку n-арное отношение является единственной родовой структурой данных, хранящихся в реляционной базе данных. Это отражено и в общем названии подхода – термин реляционный (relational) происходит от relation (отношение). Однако сам термин отношение является исключительно неточным, поскольку, говоря про любые сохраняемые данные, мы должны иметь в виду тип этих данных, значения этого типа и переменные, в которых сохраняются значения. Соответственно, для уточнения термина отношение выделяются понятия заголовка отношения, значения отношения и переменной отношения. Кроме того, нам потребуется вспомогательное понятие кортежа.

Итак, заголовком (или схемой) отношения r (Hr) называется конечное множество упорядоченных пар вида <A, T>, где A называется именем атрибута, а T обозначает имя некоторого базового типа или ранее определенного домена. По

определению требуется, чтобы все имена атрибутов в заголовке отношения были различны. В примере на рис 2 заголовком отношения СЛУЖАЩИЕ является множество пар {<слу_номер, номера_пропусков>, <слу_имя, имена>, <слу_зарп, размеры_выплат>, <слу_отд_номер, номера_отделов>}.

Если все атрибуты заголовка отношения определены на разных доменах, то, чтобы не плодить лишних имен, разумно использовать для именования атрибутов имена соответствующих доменов (не забывая, конечно, о том, что это всего лишь удобный способ именования, который не устраняет различия между понятиями домена и атрибута).

Кортежем tr, соответствующим заголовку Hr, называется множество упорядоченных триплетов вида <A, T, v>, по одному такому триплету для каждого атрибута в Hr. Третий элемент – v – триплета <A, T, v> должен являться допустимым значением типа данных или домена T. Заголовку отношения СЛУЖАЩИЕ соответствуют, например, следующие кортежи: {<слу_номер, номера_пропусков, 2934>, <слу_имя, имена, Иванов>, <слу_зарп, размеры_выплат, 22.000>, <слу_отд_номер, номера_отделов, 310>}, {<слу_номер, номера_пропусков, 2940>, <слу_имя, имена, Кузнецов>, <слу_зарп, размеры_выплат, 35.000>, <слу_отд_номер, номера_отделов, 320>}.

Телом Br отношения r называется произвольное множество кортежей tr. Одно из возможных тел отношения СЛУЖАЩИЕ показано на рис 2. Заметим, что в общем случае, как это демонстрируют, в частности, рис 2 и пример предыдущего абзаца, могут существовать такие кортежи tr, которые соответствуют Hr, но не входят в Br.

Значением Vr отношения r называется пара множеств Hr и Br. Одно из допустимых значений отношения СЛУЖАЩИЕ показано на рис 2.

В изменчивой реляционной базе данных хранятся отношения, значения которых изменяются во времени. Переменной VARr называется именованный контейнер, который может содержать любое допустимое значение Vr. Естественно,

что при определении любой VARr требуется указывать соответствующий заголовок отношения Hr.

Здесь стоит подчеркнуть, что любая принятая на практике операция обновления базы данных – INSERT (вставка кортежа в переменную отношения), DELETE (удаление кортежа из значения-отношения переменой отношения) и UPDATE (модификация кортежа значения-отношения переменной отношения) – с модельной точки зрения является операцией присваивания переменной отношения некоторого нового значения-отношения. Это совсем не означает, что перечисленные операции должны выполняться именно таким образом в СУБД: главное, чтобы результат операций соответствовал этой модельной семантике.

Заметим, что в дальнейшем в тех случаях, когда точный смысл термина понятен из контекста, мы будем использовать термин отношение как в смысле значение отношения, так и в смысле переменная отношения.

По определению, степенью, или «арностью», заголовка отношения, кортежа, соответствующего этому заголовку, тела отношения, значения отношения и переменной отношения является мощность заголовка отношения. Например, степень отношения СЛУЖАЩИЕ равна четырем, т. е. оно является 4-арным (кватернарным).

При приведенных определениях разумно считать схемой реляционной базы данных набор пар <имя_VARr, Hr>, включающий имена и заголовки всех переменных отношения, которые определены в базе данных. Реляционная база данных – это набор пар <VARr, Hr> (конечно, каждая переменная отношения в любой момент времени содержит некоторое значение-отношение, в частности, пустое).

Заметим, что в классических реляционных базах данных после определения схемы базы данных могли изменяться только значения переменных отношений. Однако теперь в большинстве реализаций допускается и изменение схемы базы данных: определение новых и изменение заголовков существующих

переменных отношений. Это принято называть эволюцией схемы базы данных.

Первичный ключ и интуитивная интерпретация реляционных понятий

По определению, первичным ключом переменной отношения является такое подмножество5) S множества атрибутов ее заголовка, что в любое время значение первичного ключа (составное, если в состав первичного ключа входит более одного атрибута) в любом кортеже тела отношения отличается от значения первичного ключа в любом другом кортеже тела этого отношения, а никакое собственное подмножество6) S этим свойством не обладает. В следующем разделе мы покажем, что существование первичного ключа у любого значения отношения является следствием одного из фундаментальных свойств отношений, а именно того свойства, что тело отношения является множеством кортежей.

Обычным житейским представлением отношения является таблица, заголовком которой является схема отношения, а строками – кортежи отношения-экземпляра; в этом случае имена атрибутов соответствуют именам столбцов данной таблицы. Поэтому иногда говорят про «столбцы таблицы», имея в виду «атрибуты отношения».

Конечно, это достаточно грубая терминология, поскольку у обычных таблиц и строки, и столбцы упорядочены, тогда как атрибуты и кортежи отношений являются элементами неупорядоченных множеств. Тем не менее, когда мы перейдем к рассмотрению практических вопросов организации реляционных баз данных и средств управления, то будем использовать эту «житейскую» терминологию. Подобной терминологии придерживаются в большинстве коммерческих реляционных СУБД. Иногда также используются термины файл как аналог таблицы, запись как аналог строки и поле как аналог столбца.

Лекция 4 «Базисные средства манипулирования реляционными данными: реляционная алгебра Кодда»

В предыдущей лекции упоминались три составляющих реляционной модели данных. Две из них – структурную и целостную части – мы рассмотрели более или менее подробно, а манипуляционной части реляционной модели данных посвящается эта и следующие две лекции. Мы уделяем данной теме такое большое внимание, поскольку понимание формальных механизмов манипулирования реляционными данными исключительно важно для понимания технологии баз данных вообще. В этой лекции после небольшого введения будет рассмотрен вариант реляционной алгебры Кодда, предложенный Кристофером Дейтом около 15 лет тому назад. Мне этот вариант алгебры кажется наиболее понятным, хотя предлагаемый набор операций несколько избыточен. В следующей лекции мы обсудим новый «минимальный» вариант алгебры, предложенный Дейтом и Дарвеном в конце 1990-х гг. Возможно, новая алгебра не очень практична, но зато красива и элегантна. После этого в лекции 6 мы перейдем к реляционному исчислению, достаточно подробно рассмотрим один из вариантов реляционного исчисления кортежей и кратко обсудим особенности исчисления доменов.

Как мы отмечали в предыдущей лекции, в манипуляционной составляющей реляционной модели данных определяются два базовых механизма манипулирования реляционными данными – основанная на теории множеств реляционная алгебра и базирующееся на математической логике (точнее, на исчислении предикатов первого порядка) реляционное исчисление. В свою очередь, обычно выделяются два вида реляционного исчисления – исчисление кортежей и исчисление доменов.

Все эти механизмы обладают одним важным свойством: они замкнуты относительно понятия отношения. Это означает, что выражения реляционной алгебры и формулы реляционного исчисления определяются над отношениями реляционных БД и

результатом их «вычисления» также являются отношения (конечно, здесь имеются в виду значения-отношения). В результате любое выражение или формула могут интерпретироваться как отношения, что позволяет использовать их в других выражениях или формулах.

Как мы увидим, алгебра и исчисление обладают большой выразительной мощностью: очень сложные запросы к базе данных могут быть выражены с помощью одного выражения реляционной алгебры или одной формулы реляционного исчисления. Именно по этой причине такие механизмы включены в реляционную модель данных. Конкретный язык манипулирования реляционными БД называется реляционно-полным, если любой запрос, формулируемый с помощью одного выражения реляционной алгебры или одной формулы реляционного исчисления, может быть сформулирован с помощью одного оператора этого языка.

Известно (и мы не будем это доказывать), что механизмы реляционной алгебры и реляционного исчисления эквивалентны, т. е. для любого допустимого выражения реляционной алгебры можно построить эквивалентную (т. е. производящую такой же результат) формулу реляционного исчисления и наоборот. Почему же в реляционной модели данных присутствуют оба эти механизма?

Дело в том, что они различаются уровнем процедурности. Выражения реляционной алгебры строятся на основе алгебраических операций (высокого уровня), и подобно тому, как интерпретируются арифметические и логические выражения, выражение реляционной алгебры также имеет процедурную интерпретацию. Другими словами, запрос, представленный на языке реляционной алгебры, может быть вычислен на основе выполнения элементарных алгебраических операций с учетом их приоритетности и возможного наличия скобок. Для формулы реляционного исчисления однозначная вычислительная интерпретация, вообще говоря, отсутствует. Формула только ставит условия, которым должны удовлетворять кортежи результирующего отношения. Поэтому

языки реляционного исчисления являются в большей степени непроцедурными, или декларативными.

Поскольку механизмы реляционной алгебры и реляционного исчисления эквивалентны, в конкретной ситуации для проверки степени реляционности некоторого языка БД можно пользоваться любым из этих механизмов.

Заметим, что крайне редко алгебра или исчисление принимается в качестве полной основы какого-либо языка БД. Обычно (например, в случае языка SQL) язык основывается на некоторой смеси алгебраических и логических конструкций. Тем не менее, знание алгебраических и логических основ языков баз данных часто применяется на практике.

Для экономии времени и места мы не будем вводить какие-либо строгие синтаксические конструкции, а в основном ограничимся рассмотрением материала на содержательном уровне.

4.1 Обзор реляционной алгебры Кодда

Основная идея реляционной алгебры состоит в том, что коль скоро отношения являются множествами, средства манипулирования отношениями могут базироваться на традиционных теоретико-множественных операциях, дополненных некоторыми специальными операциями, специфичными для реляционных баз данных.

Существует много подходов к определению реляционной алгебры, которые различаются наборами операций и способами их интерпретации, но, в принципе, являются более или менее равносильными. В данном разделе мы опишем немного расширенный начальный вариант алгебры, который был предложен Коддом (будем называть ее «алгеброй Кодда»). В этом варианте набор основных алгебраических операций состоит из восьми операций, которые делятся на два класса – теоретико-множественные операции и специальные реляционные операции. В состав теоретико-множественных операций входят операции:

1. объединения отношений;
2. пересечения отношений;

3. взятия разности отношений;
4. взятия декартова произведения отношений.

Специальные реляционные операции включают:

1. ограничение отношения;
2. проекцию отношения;
3. соединение отношений;
4. деление отношений.

Кроме того, в состав алгебры включается операция присваивания, позволяющая сохранить в базе данных результаты вычисления алгебраических выражений, и операция переименования атрибутов, дающая возможность корректно сформировать заголовок (схему) результирующего отношения.

4.2 Общая интерпретация реляционных операций

Если не вдаваться в некоторые тонкости, которые мы рассмотрим в следующих разделах, то почти для всех операций предложенного выше набора имеется очевидная и простая интерпретация.

При выполнении операции объединения (UNION) двух отношений с одинаковыми заголовками производится отношение, включающее все кортежи, которые входят хотя бы в одно из отношений-операндов.

Операция пересечения (INTERSECT) двух отношений с одинаковыми заголовками производит отношение, включающее все кортежи, которые входят в оба отношения-операнда.

Отношение, являющееся разностью (MINUS) двух отношений с одинаковыми заголовками, включает все кортежи, входящие в отношение-первый операнд, такие, что ни один из них не входит в отношение, которое является вторым операндом.

При выполнении декартова произведения (TIMES) двух отношений, пересечение заголовков которых пусто, производится отношение, кортежи которого производятся путем объединения кортежей первого и второго операндов.

Результатом ограничения (WHERE) отношения по некоторому условию является отношение, включающее кортежи отношения-операнда, удовлетворяющее этому условию.

При выполнении проекции (PROJECT) отношения на заданное подмножество множества его атрибутов производится отношение, кортежи которого являются соответствующими подмножествами кортежей отношения-операнда.

При соединении (JOIN) двух отношений по некоторому условию образуется результирующее отношение, кортежи которого производятся путем объединения кортежей первого и второго отношений и удовлетворяют этому условию.

У операции реляционного деления (DIVIDE BY) два операнда – бинарное и унарное отношения. Результирующее отношение состоит из унарных кортежей, включающих значения первого атрибута кортежей первого операнда таких, что множество значений второго атрибута (при фиксированном значении первого атрибута) включает множество значений второго операнда.

Операция переименования (RENAME) производит отношение, тело которого совпадает с телом операнда, но имена атрибутов изменены.

Операция присваивания (:=) позволяет сохранить результат вычисления реляционного выражения в существующем отношении БД.

Поскольку результатом любой реляционной операции (кроме операции присваивания, которая не вырабатывает значения) является некое отношение, можно образовывать реляционные выражения, в которых вместо отношения-операнда некоторой реляционной операции находится вложенное реляционное выражение. В построении реляционного выражения могут участвовать все реляционные операции, кроме операции присваивания. Вычислительная интерпретация реляционного выражения диктуется установленными приоритетами операций:

RENAME ≥WHERE = PROJECT ≥TIMES = JOIN = INTERSECT = DIVIDE BY ≥UNION = MINUS

В другой форме приоритеты операций показаны на рис 3. Вычисление выражения производится слева направо с учетом приоритетов операций и скобок.

Операция	Приоритет
RENAME	4
WHERE	3
PROJECT	3
TIMES	2
JOIN	2
INTERSECT	2
DIVIDE BY	2
UNION	1
MINUS	1

Рисунок 3. Таблица приоритетов операций традиционной реляционной алгебры

Замкнутость реляционной алгебры и операция переименования

Как мы отмечали в предыдущей лекции, каждое значение-отношение характеризуется заголовком (или схемой) и телом (или множеством кортежей). Поэтому, если нам действительно нужна алгебра, операции которой замкнуты относительно понятия отношения, то каждая операция должна производить отношение в полном смысле, т. е. оно должно обладать и телом, и заголовком. Только в этом случае можно будет строить вложенные выражения.

Заголовок отношения представляет собой множество пар <имя-атрибута, имя-домена>. Если посмотреть на общий обзор реляционных операций, приведенный в предыдущем подразделе, то видно, что домены атрибутов результирующего отношения однозначно определяются доменами отношений-

операндов. Однако с именами атрибутов результата не всегда все так просто.

Например, представим себе, что у отношений-операндов операции декартова произведения имеются одноименные атрибуты с одинаковыми доменами. Каким был бы заголовок результирующего отношения? Поскольку это множество, в нем не должны содержаться одинаковые элементы. Но и потерять атрибут в результате недопустимо. А это значит, что в таком случае вообще невозможно корректно выполнить операцию декартова произведения.

Аналогичные проблемы могут возникать и в случаях других двуместных операций. Для разрешения проблем в число операций реляционной алгебры вводится операция переименования. Ее следует применять в том случае, когда возникает конфликт именования атрибутов в отношениях-операндах одной реляционной операции. Тогда к одному из операндов сначала применяется операция переименования, а затем основная операция выполняется уже без всяких проблем. Более строго мы определим операцию переименования в следующей лекции, а пока лишь заметим, что результатом этой операции является отношение, совпадающее во всем с отношением-операндом, кроме того, что имя указанного атрибута изменено на заданное имя.

В дальнейшем изложении мы будем предполагать применение операции переименования во всех конфликтных ситуациях. Заметим, кстати, что невозможность применения некоторых операций к произвольным парам значений отношений без предварительного переименования атрибутов отношений операндов означает, что «алгебра» Кодда не является алгеброй отношений в математическом смысле. Описываемая в следующей главе Алгебра А такими недостатками не обладает: результотатом применения любой операции к любым отношениям является некоторое отношение.

4.3. Особенности теоретико-множественных операций реляционной алгебры

Хотя в основе теоретико-множественной части реляционной алгебры Кодда лежит классическая теория множеств, соответствующие операции реляционной алгебры обладают некоторыми особенностями.

Операции объединения, пересечения, взятия разности. Совместимость по объединению

Начнем с операции объединения отношений (все, что будет сказано по поводу объединения, верно и для операций пересечения и взятия разности отношений). Смысл операции объединения в реляционной алгебре в целом остается теоретико-множественным. Еще раз напомним (см. рис. 3.4), что в теории множеств:

- результатом объединения двух множеств A{a} и B{b} является такое множество C{c}, что для каждого c либо существует такой элемент a, принадлежащий множеству A, что c=a, либо существует такой элемент b, принадлежащий множеству B, что c=b;
- пересечением множеств A и B является такое множество C{c}, что для любого c существуют такие элементы a, принадлежащий множеству A, и b, принадлежащий множеству B, что c=a=b;
- разностью множеств A и B является такое множество C{c}, что для любого c существует такой элемент a, принадлежащий множеству A, что c=a, и не существует такой элемент b, принадлежащий B, что c=b.

Но если в теории множеств операция объединения осмысленна для любых двух множеств-операндов, то в случае реляционной алгебры результатом операции объединения должно являться отношение. Если в реляционной алгебре допустить возможность теоретико-множественного объединения двух произвольных отношений (с разными заголовками), то, конечно, результатом операции будет множество, но множество разнотипных кортежей, т. е. не отношение. Если исходить из требования замкнутости

реляционной алгебры относительно понятия отношения, то такая операция объединения является бессмысленной.

Эти соображения подводят к понятию совместимости отношений по объединению: два отношения совместимы по объединению в том и только в том случае, когда обладают одинаковыми заголовками. В развернутой форме это означает, что в заголовках обоих отношений содержится один и тот же набор имен атрибутов, и одноименные атрибуты определены на одном и том же домене (эта развернутая формулировка, вообще говоря, является излишней, но она пригодится нам в следующем абзаце).

Если два отношения совместимы по объединению, то при обычном выполнении над ними операций объединения, пересечения и взятия разности результатом операции является отношение с корректно определенным заголовком, совпадающим с заголовком каждого из отношений-операндов. Напомним, что если два отношения «почти» совместимы по объединению, т. е. совместимы во всем, кроме имен атрибутов, то до выполнения операции типа объединения эти отношения можно сделать полностью совместимыми по объединению путем применения операции переименования.

Для иллюстрации операций объединения, пересечения и взятия разности предположим, что в базе данных имеются два отношения СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 и СЛУЖАЩИЕ_В_ПРОЕКТЕ_2 с одинаковыми схемами {СЛУ_НОМЕР, СЛУ_ИМЯ, СЛУ_ЗАРП, СЛУ_ОТД_НОМЕР} (имена доменов опущены по причине очевидности). Каждое из отношений содержит данные о служащих, участвующих в соответствующем проекте. На рис 4. показано примерное наполнение каждого из двух отношений (некоторые служащие участвуют в обоих проектах).

СЛУЖАЩИЕ_В_ПРОЕКТЕ_1

СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_НОМЕР
2934	Иванов	22000.00	310
2935	Петров	30000.00	310
2936	Сидоров	18000.00	313
2937	Федоров	20000.00	310
2938	Иванова	22000.00	315

СЛУЖАЩИЕ_В_ПРОЕКТЕ_2

СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_НОМЕР
2934	Иванов	22000.00	310
2935	Петров	30000.00	310
2939	Сидоренко	18000.00	313
2940	Федоренко	20000.00	310
2941	Иваненко	22000.00	315

Рисунок 4. Примерное наполнение отношений СЛУЖАЩИЕ _В_ПРОЕКТЕ_1 и СЛУЖАЩИЕ _В_ПРОЕКТЕ_2

Тогда выполнение операции СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 UNION СЛУЖАЩИЕ_В_ПРОЕКТЕ_2 позволит получить информацию обо всех служащих, участвующих в обоих проектах. Выполнение операции СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 INTERSECT СЛУЖАЩИЕ_В_ПРОЕКТЕ_2 позволит получить данные о служащих, которые одновременно участвуют в двух проектах. Наконец, операция СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 MINUS СЛУЖАЩИЕ_В_ПРОЕКТЕ_2 выработает отношение, содержащее кортежи служащих, которые участвуют только в первом проекте. Результаты этих операций показаны на рис 5.

СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 UNION **СЛУЖАЩИЕ_В_ПРОЕКТЕ_2**

СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_НОМЕР
2934	Иванов	22000.00	310
2935	Петров	30000.00	310
2939	Сидоренко	18000.00	313
2940	Федоренко	20000.00	310
2941	Иваненко	22000.00	315
2936	Сидоров	18000.00	313
2937	Федоров	20000.00	310
2938	Иванова	22000.00	315

СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 INTERSECT **СЛУЖАЩИЕ_В_ПРОЕКТЕ_2**

СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_НОМЕР
2934	Иванов	22000.00	310
2935	Петров	30000.00	310

СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 MINUS **СЛУЖАЩИЕ_В_ПРОЕКТЕ_2**

СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_НОМЕР
2936	Сидоров	18000.00	313
2937	Федоров	20000.00	310
2938	Иванова	22000.00	315

Рисунок 5. Результаты выполнения операций UNION, INTERSECT и MINUS

Заметим, что включение в состав операций реляционной алгебры трех операций объединения, пересечения и взятия разности является, очевидно, избыточным, поскольку, например, операция пересечения выражается через операцию взятия разности14). Тем не менее Кодд в свое время решил включить все три операции, исходя из интуитивных потребностей далекого от математики потенциального пользователя системы реляционных БД.

Операция расширенного декартова произведения и совместимость отношений относительно этой операции

Другие проблемы связаны с операцией взятия декартова произведения двух отношений. В теории множеств декартово произведение может быть получено для любых двух множеств, и элементами результирующего множества являются пары, составленные из элементов первого и второго множеств. Если говорить более точно, декартовым произведением множеств A{a} и B{b} является такое множество пар C{<c1, 2>}, что для каждого элемента <c1, c2> множества C существуют такой элемент a множества A, что c1=a, и такой элемент b множества B, что c2=b.

Поскольку отношения являются множествами, для любых двух отношений возможно получение прямого произведения. Но результат не будет отношением! Элементами результата будут не кортежи, а пары кортежей.

Поэтому в реляционной алгебре используется специализированная форма операции взятия декартова произведения – расширенное декартово произведение отношений. При взятии расширенного декартова произведения двух отношений элементом результирующего отношения является кортеж, который представляет собой объединение одного кортежа первого отношения и одного кортежа второго отношения.

Приведем более точное определение операции расширенного декартова произведения. Пусть имеются два отношения R1{a1, a2, …, an} и R2{b1, b2, …, bm}. Тогда результатом операции R1 TIMES R2 является отношение R{a1, a2, …, an, b1, b2, …, bm}, тело которого является множеством кортежей вида {ra1, ra2, …, ran, rb1, rb2, …, rbm} таких, что {ra1, ra2, …, ran} входит в тело R1, а {rb1, rb2, …, rbm} входит в тело R2.

Но теперь возникает вторая проблема – как получить корректно сформированный заголовок отношения-результата? Поскольку схема результирующего отношения является объединением схем отношений-операндов, то очевидной

проблемой может быть именование атрибутов результирующего отношения, если отношения-операнды обладают одноименными атрибутами.

Эти соображения приводят к введению понятия совместимости по взятию расширенного декартова произведения. Два отношения совместимы по взятию расширенного декартова произведения в том и только в том случае, если пересечение множеств имен атрибутов, взятых из их схем отношений, пусто. Любые два отношения всегда могут стать совместимыми по взятию декартова произведения, если применить операцию переименования к одному из этих отношений.

Для наглядности предположим, что в придачу к введенным ранее отношениям СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 и СЛУЖАЩИЕ_В_ПРОЕКТЕ_2 в базе данных содержится еще и отношение ПРОЕКТЫ со схемой {ПРОЕКТ_НАЗВ, ПРОЕКТ_РУК} (имена доменов снова опущены) и телом, показанным на рис 6. На этом же рисунке показан результат операции СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 TIMES ПРОЕКТЫ.

Следует заметить, что операция взятия декартова произведения не является слишком осмысленной на практике. Во-первых, мощность тела ее результата очень велика даже при допустимых мощностях операндов, а во-вторых, результат операции не более информативен, чем взятые в совокупности операнды. Как будет показано далее, основной смысл включения операции расширенного декартова произведения в состав реляционной алгебры Кодда состоит в том, что на ее основе определяется действительно полезная операция соединения.

По поводу теоретико-множественных операций реляционной алгебры следует еще заметить, что все четыре операции являются ассоциативными. Т. е. если обозначить через OP любую из четырех операций, то (A OP B) OP C = A OP (B OP C), и, следовательно, без внесения двусмысленности можно писать A OP B OP C (A, B и C – отношения, обладающие свойствами, необходимыми для корректного выполнения

соответствующей операции). Все операции, кроме взятия разности, являются коммутативными, т. е. A OP B = B OP A.

ПРОЕКТЫ

ПРОЕКТ_НАЗВ	ПРОЕКТ_РУК
ПРОЕКТ 1	Иванов
ПРОЕКТ 2	Иваненко

СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 TIMES **ПРОЕКТЫ**

СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_НОМЕР	ПРОЕКТ_НАЗВ	ПРОЕКТ_РУК
2934	Иванов	22000.00	310	ПРОЕКТ 1	Иванов
2935	Петров	30000.00	310	ПРОЕКТ 1	Иванов
2936	Сидоров	18000.00	313	ПРОЕКТ 1	Иванов
2937	Федоров	20000.00	310	ПРОЕКТ 1	Иванов
2938	Иванова	22000.00	315	ПРОЕКТ 1	Иванов
2934	Иванов	22000.00	310	ПРОЕКТ 2	Иваненко
2935	Петров	30000.00	310	ПРОЕКТ 2	Иваненко
2936	Сидоров	18000.00	313	ПРОЕКТ 2	Иваненко
2937	Федоров	20000.00	310	ПРОЕКТ 2	Иваненко
2938	Иванова	22000.00	315	ПРОЕКТ 2	Иваненко

Рисунок 6. Отношение ПРОЕКТЫ и результат операции СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 TIMES ПРОЕКТЫ

Лекция № 5 «Базисные средства манипулирования реляционными данными: реляционное исчисление»

Предположим, что мы работаем с базой данных, которая состоит из отношений СЛУЖАЩИЕ {СЛУ_НОМ, СЛУ_ИМЯ, СЛУ_ЗАРП, ПРО_НОМ} и ПРОЕКТЫ {ПРО_НОМ, ПРОЕКТ_РУК, ПРО_ЗАРП} (в отношении ПРОЕКТЫ атрибут ПРОЕКТ_РУК содержит имена служащих, являющихся руководителями проектов, а атрибут ПРО_ЗАРП – среднее

значение зарплаты, получаемой участниками проекта), и хотим узнать имена и номера служащих, которые являются руководителями проектов со средней заработной платой, превышающей 18000 руб.

Если бы для формулировки такого запроса использовалась реляционная алгебра, то мы получили бы, например, следующее алгебраическое выражение:

(СЛУЖАЩИЕ JOIN ПРОЕКТЫ WHERE (СЛУ_ИМЯ = ПРОЕКТ_РУК AND

ПРО_ЗАРП > 18000.00)) PROJECT (СЛУ_ИМЯ, СЛУ_НОМ)

Это выражение можно было бы прочитать, например, следующим образом:

- выполнить эквисоединение отношений СЛУЖАЩИЕ и ПРОЕКТЫ по условию СЛУ_ИМЯ = ПРОЕКТ_РУК;
- ограничить полученное отношение по условию ПРО_ЗАРП > 18000.00;
- спроецировать результат предыдущей операции на атрибут СЛУ_ИМЯ, СЛУ_НОМ.

Мы четко сформулировали последовательность шагов выполнения запроса, каждый из которых соответствует одной реляционной операции.

Если же сформулировать тот же запрос с использованием реляционного исчисления, которому посвящается эта лекция, то мы получили бы два определения переменных:

RANGE СЛУЖАЩИЙ IS СЛУЖАЩИЕ и

RANGE ПРОЕКТ IS ПРОЕКТЫ

и выражение

СЛУЖАЩИЙ.СЛУ_ИМЯ, СЛУЖАЩИЙ.СЛУ_НОМ WHERE EXISTS (СЛУЖАЩИЙ.СЛУ_ИМЯ = ПРОЕКТ.ПРОЕКТ_РУК AND ПРОЕКТ.ПРО_ЗАРП > 18000.00).

Это выражение можно было бы прочитать, например, следующим образом: выдать значения СЛУ_ИМЯ и СЛУ_НОМ для каждого кортежа служащих такого, что существует кортеж проектов со значением ПРОЕКТ_РУК, совпадающим со

значением СЛУ_НОМ этого кортежа служащих, и значением ПРО_ЗАРП, большим 18000.00.

Во второй формулировке мы указали лишь характеристики результирующего отношения, но ничего не сказали о способе его формирования. В этом случае система сама должна решить, какие операции и в каком порядке нужно выполнить над отношениями СЛУЖАЩИЕ и ПРОЕКТЫ. Обычно говорят, что алгебраическая формулировка является процедурной, т. е. задающей последовательность действий для выполнения запроса, а логическая – описательной (или декларативной), поскольку она всего лишь описывает свойства желаемого результата. Как мы указывали в начале лекции 4, на самом деле эти два механизма эквивалентны, и существуют не слишком сложные правила преобразования одного формализма в другой.

Реляционное исчисление является прикладной ветвью формального механизма исчисления предикатов первого порядка25). В основе исчисления лежит понятие переменной с определенной для нее областью допустимых значений и понятие правильно построенной формулы, опирающейся на переменные, предикаты и кванторы.

В зависимости от того, что является областью определения переменной, различают исчисление кортежей и исчисление доменов. В исчислении кортежей областями определения переменных являются тела отношений базы данных, т. е. допустимым значением каждой переменной является кортеж тела некоторого отношения. В исчислении доменов областями определения переменных являются домены, на которых определены атрибуты отношений базы данных, т. е. допустимым значением каждой переменной является значение некоторого домена. Мы рассмотрим более подробно исчисление кортежей, а в конце лекции коротко опишем особенности исчисления доменов.

Как и в лекциях, посвященных реляционной алгебре, в этой лекции нам не удастся избежать использования некоторого конкретного синтаксиса, который мы тем не менее формально

определять не будем. Те или иные синтаксические конструкции будут вводиться по мере необходимости. В совокупности используемый синтаксис близок, но не полностью совпадает с синтаксисом языка баз данных QUEL, который долгое время являлся основным языком известной реляционной СУБД Ingres.

5.1. Исчисление кортежей

Для определения кортежной переменной используется оператор RANGE. Например, для того чтобы определить переменную СЛУЖАЩИЙ, областью определения которой является отношение СЛУЖАЩИЕ, нужно употребить конструкцию

RANGE СЛУЖАЩИЙ IS СЛУЖАЩИЕ

Как уже говорилось, из этого определения следует, что в любой момент времени переменная СЛУЖАЩИЙ представляет некоторый кортеж отношения СЛУЖАЩИЕ. При использовании кортежных переменных в формулах можно ссылаться на значение атрибута переменной (это аналогично тому, как, например, при программировании на языке С можно сослаться на значение поля структурной переменной). Например, для того, чтобы сослаться на значение атрибута СЛУ_ИМЯ переменной СЛУЖАЩИЙ, нужно употребить конструкцию СЛУЖАЩИЙ.СЛУ_ИМЯ.

Правильно построенные формулы

Правильно построенная формула (Well-Formed Formula, WFF) служит для выражения условий, накладываемых на кортежные переменные.

Простые условия

Основой WFF являются простые условия, представляющие собой операции сравнения скалярных значений (значений атрибутов переменных или литерально заданных констант). Например, конструкции

СЛУЖАЩИЙ.СЛУ_НОМ = 2934 и

СЛУЖАЩИЙ.СЛУ_НОМ = ПРОЕКТ.ПРОЕКТ_РУК

являются простыми условиями. Первое условие принимает значение true в том и только в том случае, когда значение атрибута СЛУ_НОМ кортежной переменной

СЛУЖАЩИЙ равно 2934. Второе условие принимает значение true в том и только в том случае, когда значения атрибутов СЛУ_НОМ и ПРОЕКТ_РУК переменных СЛУЖАЩИЙ и ПРОЕКТ совпадают.

По определению, простое сравнение является WFF, а WFF, заключенная в круглые скобки, представляет собой простое сравнение.

Более сложные варианты WFF строятся с помощью логических связок NOT, AND, OR и IF ... THEN26) с учетом обычных приоритетов операций (NOT > AND > OR) и возможности расстановки скобок. Так, если form – WFF, а comp – простое сравнение, то NOT form, comp AND form, comp OR form и IF comp THEN form являются WFF.

Для примеров воспользуемся отношениями СЛУЖАЩИЕ, ПРОЕКТЫ и НОМЕРА_ПРОЕКТОВ из предыдущей лекции (см.рис 7).

Правильно построенной является следующая формула:

IF СЛУЖАЩИЙ.СЛУ_ИМЯ = 'Иванов'

THEN (СЛУЖАЩИЙ.СЛУ_ЗАРП >= 22400.00 AND СЛУЖАЩИЙ.ПРО_НОМ = 1)

Эта формула будет принимать значение true для следующих значений кортежной переменной СЛУЖАЩИЙ:

СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_НОМ
2934	Иванов	22400.00	1
2935	Петров	29600.00	1
2936	Сидоров	18000.00	1
2937	Федоров	20000.00	1
2938	Иванова	22000.00	1
2935	Петров	29600.00	2
2939	Сидоренко	18000.00	2
2940	Федоренко	20000.00	2
2941	Иваненко	22000.00	2

СЛУЖАЩИЕ

СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_НОМ
2934	Иванов	22400.00	1
2935	Петров	29600.00	1
2936	Сидоров	18000.00	1
2937	Федоров	20000.00	1
2938	Иванова	22000.00	1
2934	Иванов	22400.00	2
2935	Петров	29600.00	2
2939	Сидоренко	18000.00	2
2940	Федоренко	20000.00	2
2941	Иваненко	22000.00	2

ПРОЕКТЫ

ПРО_НОМ	ПРОЕКТ_РУК
1	Иванов
2	Иваненко

НОМЕРА_ПРОЕКТОВ

ПРО_НОМ
1
2

Рисунок 7 Примерные значения отношений СЛУЖАЩИЕ, ПРОЕКТЫ и НОМЕРА_ПРОЕКТОВ

Конечно, нужно представлять себе какой-нибудь способ реализации системы, которая сможет по заданной WFF при существующем состоянии базы данных произвести такой результат. И таким очевидным способом является следующий: в некотором порядке просмотреть область определения переменной и к каждому очередному кортежу применить условие. Результатом будет то множество кортежей, для которых при вычислении условия производится значение true. Очевидно, что результат эквивалентен выполнению алгебраической операции СЛУЖАЩИЕ WHERE (NOT

(СЛУЖАЩИЙ.СЛУ_ИМЯ = 'Иванов') OR (СЛУЖАЩИЙ.СЛУ_ЗАРП >= 22400.00 AND СЛУЖАЩИЙ.ПРО_НОМ = 1) над отношением, тело которого представляет собой область определения кортежной переменной.

Пусть имеется следующее определение кортежной переменной ПРОЕКТ:

RANGE ПРОЕКТ IS ПРОЕКТЫ

Вот еще пример правильно построенной формулы:

СЛУЖАЩИЙ.СЛУ_ИМЯ = ПРОЕКТ.ПРОЕКТ_РУК

Эта формула будет принимать значение true для следующих пар значений кортежных переменных СЛУЖАЩИЙ и ПРОЕКТ:

СЛУЖАЩИЕ				ПРОЕКТЫ	
СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_НОМ	ПРО_НОМ	ПРОЕКТ_РУК
2934	Иванов	22400.00	1	1	Иванов
2941	Иваненко	22000.00	2	2	Иваненко
2934	Иванов	22400.00	2	1	Иванов

Очевидный способ реализации системы, которая по заданной WFF при существующем состоянии базы данных производит такой результат, заключается в следующем. В некотором порядке просматривать область определения (например) переменной СЛУЖАЩИЙ. Для каждого текущего кортежа из области определения переменной СЛУЖАЩИЙ просматривать область определения переменной ПРОЕКТ. Оставлять в области истинности те пары кортежей, для которых формула принимает значение true. Возможен и альтернативный подход: начать просмотр с области определения переменной ПРОЕКТ, и для каждого кортежа ПРОЕКТ просматривать область определения СЛУЖАЩИЙ.

Здесь нужно сделать несколько замечаний. Во-первых, если бы в данном случае формула была тождественно истинной (например, имела вид

(СЛУЖАЩИЙ.СЛУ_ИМЯ = СЛУЖАЩИЙ.СЛУ_ИМЯ)

AND (ПРОЕКТ.ПРОЕКТ_РУК = ПРОЕКТ.ПРОЕКТ_РУК))

то областью истинности этой формулы являлось бы декартово произведение (в строгом математическом смысле) тел отношений СЛУЖАЩИЙ и ПРОЕКТ. В реляционном исчислении кортежей, как и в реляционной алгебре, принято иметь дело с операцией расширенного декартова произведения, и поэтому считается, что в подобных случаях областью истинности WFF является отношение, заголовок которого представляет собой объединение заголовков отношений, на телах которых определены кортежные переменные, а кортежи являются объединением соответствующих кортежей из областей определения переменных. При этом имя атрибута результирующего отношения уточняется именем соответствующей переменной. Поэтому правильнее было бы изображать область истинности формулы

СЛУЖАЩИЙ.СЛУ_ИМЯ = ПРОЕКТ.ПРОЕКТ_РУК

следующим образом:

СЛУЖАЩИЙ. СЛУ_ИМЯ	СЛУЖАЩИЙ. СЛУ_ЗАРП	СЛУЖАЩИЙ. ПРО_НОМ	ПРОЕКТ. ПРО_НОМ
Иванов	22400.00	1	1
Иваненко	22000.00	2	2
Иванов	22400.00	2	1

СЛУЖАЩИЙ. СЛУ_НОМЕР	ПРОЕКТ. ПРОЕКТ_ РУК
2934	Иванов
2941	Иваненко
2934	Иванов

Во-вторых, как видно, показанное результирующее отношение в точности совпадает с результатом алгебраической операции СЛУЖАЩИЕ JOIN ПРОЕКТЫ WHERE СЛУ_ИМЯ = ПРОЕКТ_РУК с учетом особенности именования атрибутов результирующего отношения. Наконец, заметим, что описанный выше способ реализации, который приводит к получению области истинности рассмотренной формулы, в

действительности является наиболее общим (и зачастую неоптимальным) способом выполнения операций соединения (он называется методом вложенных циклов – nested loops join).

Кванторы, свободные и связанные переменные

При построении WFF допускается использование кванторов существования (EXISTS) и всеобщности (FORALL). Если form – это WFF, в которой участвует переменная var, то конструкции EXISTS var (form) и FORALL var (form) представляют собой WFF. По определению, формула EXISTS var (form) принимает значение true в том и только в том случае, если в области определения переменной var найдется хотя бы одно значение (кортеж), для которого WFF form принимает значение true. Формула FORALL var (form) принимает значение true, если для всех значений переменной var из ее области определения WFF form принимает значение true.

Переменные, входящие в WFF, могут быть свободными или связанными. По определению, все переменные, входящие в WFF, при построении которой не использовались кванторы, являются свободными. Фактически, это означает, что если для какого-то набора значений свободных кортежных переменных при вычислении WFF получено значение true, то эти значения кортежных переменных могут входить в результирующее отношение. Если же имя переменной использовано сразу после квантора при построении WFF вида EXISTS var (form) или FORALL var (form), то в этой WFF и во всех WFF, построенных с ее участием, var является связанной переменной. Это означает, что такая переменная не видна за пределами минимальной WFF, связавшей эту переменную. При вычислении значения такой WFF используется не одно значение связанной переменной, а вся область ее определения.

Пусть здесь и далее в этом разделе СЛУ1 и СЛУ2 представляют собой две кортежные переменные, определенные на отношении СЛУЖАЩИЕ. Тогда WFF

EXISTS СЛУ2 (СЛУ1.СЛУ_ЗАРП > СЛУ2.СЛУ_ЗАРП)

для текущего кортежа переменной СЛУ1 принимает значение true в том и только в том случае, если во всем отношении

СЛУЖАЩИЕ найдется такой кортеж (ассоциированный с переменной СЛУ2), чтобы значение его атрибута СЛУ_ЗАРП удовлетворяло внутреннему условию сравнения. Легко видеть, что эта формула принимает значение true только для тех значений кортежной переменной СЛУ1, которые соответствуют служащим, не получающим минимальную зарплату. Соответствующее множество кортежей показано на рис. 8 а.

(a) Область истинности WFF

```
EXISTS СЛУ2 (СЛУ1.СЛУ_ЗАРП > СЛУ2.СЛУ_ЗАРП)
```

СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_НОМ
2934	Иванов	22400.00	1
2935	Петров	29600.00	1
2937	Федоров	20000.00	1
2938	Иванова	22000.00	1
2934	Иванов	22400.00	2
2935	Петров	29600.00	2
2940	Федоренко	20000.00	2
2941	Иваненко	22000.00	2

(b) Область истинности WFF

```
FORALL СЛУ2 (СЛУ1.СЛУ_ЗАРП ≥ СЛУ2.СЛУ_ЗАРП)
```

СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_НОМ
2935	Петров	29600.00	1
2935	Петров	29600.00	2

Рисунок 8. Примеры правильно построенных формул с кванторами

Правильно построенная формула

FORALL СЛУ2 (СЛУ1.СЛУ_ЗАРП ≥СЛУ2.СЛУ_ЗАРП)

для текущего кортежа переменной СЛУ1 принимает значение true в том и только в том случае, если для всех кортежей отношения СЛУЖАЩИЕ (связанных с переменной СЛУ2) значения атрибута СЛУ_ЗАРП удовлетворяют условию сравнения. Снова легко видеть, что формула принимает

значение true только для тех значений кортежной переменной СЛУ1, которые соответствуют служащим, получающим максимальную зарплату. Соответствующее множество кортежей показано на рис.

Очевидно, что показанные отношения соответствуют условиям обеих формул. Но как в данном случае можно реализовать систему, которая по заданной формуле производит правильный результат? Наиболее очевидный способ интерпретации обеих обсуждавшихся выше формул следующий. В некотором порядке просматривать область определения свободной кортежной переменной СЛУ1. Для каждого очередного кортежа из области определения СЛУ1 просматривать область определения связанной переменной СЛУ2 до тех пор, пока не будет установлено истинностное значение формулы для данного кортежа СЛУ1 (в случае наличия квантора существования процесс просмотра для СЛУ2 можно остановить после нахождения первого кортежа, для которого значением подформулы, находящейся под знаком квантора, станет true; при наличии квантора всеобщности необходимо просмотреть всю область определения СЛУ2). Заметим, что здесь мы снова получаем два цикла, как и при интерпретации WFF с двумя свободными переменными. Но в данном случае во внешнем цикле обязательно просматривается область определения свободной переменной.

На самом деле, правильнее говорить не о свободных и связанных переменных, а о свободных и связанных вхождениях переменных. Если переменная var является связанной в WFF form, то во всех WFF, включающих form, вне form может использоваться вхождение того же имени переменной var, которое может быть свободным или связанным, но в любом случае не имеет никакого отношения к вхождению переменной var в WFF form. Вот пример:

EXISTS СЛУ2 (СЛУ1.ПРО_НОМ = СЛУ2.ПРО_НОМ
AND СЛУ1.СЛУ_НОМЕР = СЛУ2.СЛУ_НОМЕР)
AND FORALL СЛУ2 (IF СЛУ1.ПРО_НОМ = СЛУ2.ПРО_НОМ

THEN СЛУ1.СЛУ_ЗАРП = СЛУ2.СЛУ_ЗАРП)

Эта формула принимает значение true только для тех значений переменной СЛУ1, которые соответствуют служащим, участвующим в проектах с более чем одним участником, причем все участники проекта получают одну и ту же зарплату. Здесь мы имеем два связанных вхождения переменной СЛУ2 с совершенно разным смыслом. Грубо говоря, для текущего значения переменной СЛУ1 переменная СЛУ2 два раза «пробежит» свою область определения – первый раз при вычислении части формулы с квантором существования, а второй при вычислении части с квантором всеобщности. Кстати, к тому же результату приведет формула с одним квантором всеобщности вида:

FORALL СЛУ2 (IF (СЛУ1.ПРО_НОМ = СЛУ2.ПРО_НОМ AND

СЛУ1.СЛУ_НОМЕР ≠ СЛУ2.СЛУ_НОМЕР)

THEN СЛУ1.СЛУ_ЗАРП = СЛУ2.СЛУ_ЗАРП)

Легко заметить, что кванторы можно трактовать как булевские функции (функции, принимающие значения true или false) над множеством значений связанной кортежной переменной. С тем же успехом можно ввести в реляционное исчисление числовые функции над множествами, такие, как MIN (минимальное значение), MAX (максимальное значение), AVG (среднее значение) и т. д.

В этом случае можно было бы написать, например, WFF

СЛУ1.СЛУ_ЗАРП > MIN СЛУ2.СЛУ_ЗАРП (СЛУ1.ПРО_НОМ = СЛУ2.ПРО_НОМ)

в области истинности которой содержатся все кортежи отношения СЛУЖАЩИЕ, соответствующие тем служащим, которые получают заработную плату, превышающую минимальную зарплату служащих, участвующих в том же проекте. Понятно, что для получения результирующего отношения можно интерпретировать формулу таким же образом, как в обсуждавшемся выше случае наличия кванторов.

Целевые списки и выражения реляционного исчисления

Итак, WFF обеспечивают средства формулировки условия выборки из отношений БД. Чтобы можно было использовать исчисление для реальной работы с БД, требуется еще один компонент, который определяет набор и имена атрибутов результирующего отношения. Этот компонент называется целевым списком (target list).

Целевой список строится из целевых элементов, каждый из которых может иметь следующий вид:

1. var.attr, где var – имя свободной переменной соответствующей WFF, а attr – имя атрибута отношения, на котором определена переменная var;
2. var, что эквивалентно наличию подсписка var.attr1, var.attr2, ..., var.attrn, где {attr1, attr2, ..., attrn} включает имена всех атрибутов определяющего отношения;
3. new_name = var.attr; new_name – новое имя соответствующего атрибута результирующего отношения.

Последний вариант требуется в тех случаях, когда в WFF используется несколько свободных переменных с одинаковой областью определения. Фактически применение целевого списка к области истинности WFF эквивалентно действию алгебраической операции проекции, а последний из приведенных вариантов представляет собой некоторую разновидность алгебраической операции переименования атрибута.

Выражением реляционного исчисления кортежей называется конструкция вида target_list WHERE WFF. Значением выражения является отношение, тело которого определяется WFF, а множество атрибутов и их имена – целевым списком.

В качестве простого примера покажем выражение реляционного исчисления кортежей, результат которого совпадает с результатом операции СЛУЖАЩИЕ DIVIDE BY НОМЕРА_ПРОЕКТОВ:

СЛУ1, СЛУ2 RANGE IS СЛУЖАЩИЕ

НОМЕР_ПРОЕКТА range is НОМЕРА_ПРОЕКТОВ

СЛУ1.СЛУ_НОМЕР, СЛУ1.СЛУ_ИМЯ, СЛУ1.СЛУ_ЗАРП
WHERE FORALL НОМЕР_ПРОЕКТА EXISTS СЛУ2
(СЛУ1.СЛУ_НОМЕР = СЛУ2.СЛУ_НОМЕР AND
СЛУ1.ПРО_НОМ = НОМЕРА_ПРОЕКТОВ.ПРО_НОМ)

Конечно, результатом этого выражения является отношение

СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП
2934	Иванов	22400.00
2935	Петров	29600.00

Лекция № 6 «Проектирование реляционных баз данных на основе принципов нормализации: первые шаги нормализации»

В данной лекции речь пойдет о нормализации схем отношений с учетом только функциональных зависимостей между атрибутами отношений. Эти «первые шаги» нормализации позволяют получить схему базы данных, в которых все переменные отношений находятся в нормальной форме Бойса-Кодда, обычно расцениваемой удовлетворительной для большей части приложений.

При проектировании базы данных решаются две основные проблемы.

Каким образом отобразить объекты предметной области в абстрактные объекты модели данных, чтобы это отображение не противоречило семантике предметной области и было, по возможности, лучшим (эффективным, удобным и т. д.)? Часто эту проблему называют проблемой логического проектирования баз данных.

Как обеспечить эффективность выполнения запросов к базе данных, т. е. каким образом, имея в виду особенности конкретной СУБД, расположить данные во внешней памяти, создания каких дополнительных структур (например, индексов)

потребовать и т. д.? Эту проблему обычно называют проблемой физического проектирования баз данных.

В случае реляционных баз данных трудно предложить какие-либо общие рецепты по части физического проектирования. Здесь слишком многое зависит от используемой СУБД. Поэтому мы ограничимся вопросами логического проектирования реляционных баз данных, которые существенны при использовании любой реляционной СУБД.

Более того, мы не будем касаться очень важного аспекта проектирования – определения ограничений целостности общего вида (за исключением ограничений, задаваемых функциональными и многозначными зависимостями, а также зависимостями проекции/соединения). Дело в том, что при использовании СУБД с развитыми механизмами ограничений целостности (например, SQL-ориентированных систем) трудно предложить какой-либо универсальный подход к определению ограничений целостности. Эти ограничения могут иметь произвольно сложную форму, и их формулировка пока относится скорее к области искусства, чем инженерного мастерства. Самое большее, что предлагается по этому поводу в литературе, это автоматическая проверка непротиворечивости набора ограничений целостности.

Так что в этой и следующей лекциях мы будем считать, что проблема проектирования реляционной базы данных состоит в обоснованном принятии решений о том, из каких отношений должна состоять БД и какие атрибуты должны быть у этих отношений.

В этой и следующей лекциях будет рассмотрен классический подход, при котором весь процесс проектирования базы данных осуществляется в терминах реляционной модели данных методом последовательных приближений к удовлетворительному набору схем отношений. Исходной точкой является представление предметной области в виде одного или нескольких отношений, и на каждом шаге проектирования производится некоторый набор схем отношений, обладающих «улучшенными» свойствами. Процесс

проектирования представляет собой процесс нормализации схем отношений, причем каждая следующая нормальная форма обладает свойствами, в некотором смысле, лучшими, чем предыдущая.

Каждой нормальной форме соответствует определенный набор ограничений, и отношение находится в некоторой нормальной форме, если удовлетворяет свойственному ей набору ограничений. Примером может служить ограничение первой нормальной формы – значения всех атрибутов отношения атомарны31). Поскольку требование первой нормальной формы является базовым требованием классической реляционной модели данных, мы будем считать, что исходный набор отношений уже соответствует этому требованию.

В теории реляционных баз данных обычно 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 (5NF или PJ/NF).

Основные свойства нормальных форм состоят в следующем:

- каждая следующая нормальная форма в некотором смысле лучше предыдущей нормальной формы;
- при переходе к следующей нормальной форме свойства предыдущих нормальных форм сохраняются.

В основе процесса проектирования лежит метод нормализации, т. е. декомпозиции отношения, находящегося в предыдущей нормальной форме, на два или более отношений, которые удовлетворяют требованиям следующей нормальной формы.

В этой лекции мы обсудим первые шаги процесса нормализации, в которых учитываются функциональные зависимости между атрибутами отношений. Хотя мы и

называем эти шаги первыми, именно они имеют основную практическую важность, поскольку позволяют получить схему реляционной базы данных, в большинстве случаев удовлетворяющую потребности приложений.

6.1. Минимальные функциональные зависимости и вторая нормальная форма

Пусть имеется переменная отношения СЛУЖАЩИЕ_ПРОЕКТЫ_ЗАДАНИЯ {СЛУ_НОМ, СЛУ_УРОВ, СЛУ_ЗАРП, ПРО_НОМ, СЛУ_ЗАДАН}. Новые атрибуты СЛУ_УРОВ и СЛУ_ЗАДАН содержат, соответственно, данные о разряде служащего и о задании, которое выполняет служащий в данном проекте. Будем считать, что разряд служащего определяет размер его заработной платы и что каждый служащий может участвовать в нескольких проектах, но в каждом проекте он выполняет только одно задание. Тогда очевидно, что единственно возможным ключом отношения СЛУЖАЩИЕ_ПРОЕКТЫ_ЗАДАНИЯ является составной атрибут {СЛУ_НОМ, ПРО_НОМ}. Диаграмма минимального множества FD показана на рис 9, а возможное тело значения отношения – на рис 10.

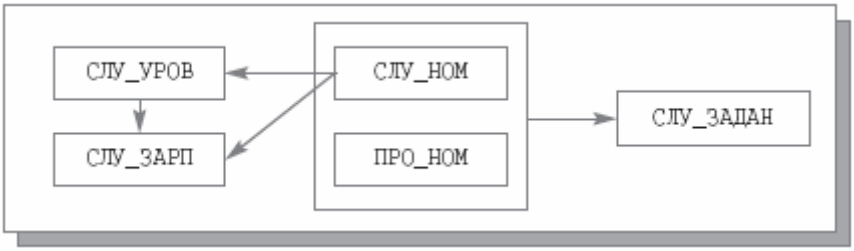

Рисунок 9. Диаграмма множества FD отношения СЛУЖАЩИЕ_ПРОЕКТЫ_ЗАДАНИЯ

СЛУ_НОМ	СЛУ_УРОВ	СЛУ_ЗАРП	ПРО_НОМ	СЛУ_ЗАДАН
2934	2	22400.00	1	A
2935	3	29600.00	1	B
2936	1	20000.00	1	C
2937	1	20000.00	1	D
2934	2	22400.00	2	D
2935	3	29600.00	2	C
2936	1	20000.00	2	B
2937	1	20000.00	2	A

Рисунок 10 Возможное значение переменной отношения СЛУЖАЩИЕ_ПРОЕКТЫ_ЗАДАНИЯ

Аномалии обновления, возникающие из-за наличия неминимальных функциональных зависимостей

Во множество FD отношения СЛУЖАЩИЕ_ПРОЕКТЫ_ЗАДАНИЯ входит много FD, в которых детерминантом является не возможный ключ отношения (соответствующие стрелки в диаграмме начинаются не с {СЛУ_НОМ, ПРО_НОМ}, т. е. некоторые функциональные зависимости атрибутов от возможного ключа не являются минимальными). Это приводит к так называемым аномалиям обновления. Под аномалиями обновления понимаются трудности, с которыми приходится сталкиваться при выполнении операций добавления кортежей в отношение (INSERT), удаления кортежей (DELETE) и модификации кортежей (UPDATE). Обсудим сначала аномалии обновления, вызываемые наличием FD СЛУ_НОМ→СЛУ_УРОВ (эти аномалии связаны с избыточностью хранения значений атрибутов СЛУ_УРОВ и СЛУ_ЗАРП в каждом кортеже, описывающем задание служащего в некотором проекте).

Добавление кортежей. Мы не можем дополнить отношение СЛУЖАЩИЕ_ПРОЕКТЫ_ЗАДАНИЯ данными о служащем, который в данное время еще не участвует ни в одном проекте (ПРО_НОМ является частью первичного ключа и не

может содержать неопределенных значений). Между тем часто бывает, что сначала служащего принимают на работу, устанавливают его разряд и размер зарплаты, а лишь потом назначают для него проект.

Удаление кортежей. Мы не можем сохранить в отношении СЛУЖАЩИЕ_ПРОЕКТЫ_ЗАДАНИЯ данные о служащем, завершившем участие в своем последнем проекте (по той причине, что значение атрибута ПРО_НОМ для этого служащего становится неопределенным). Между тем характерна ситуация, когда между проектами возникают перерывы, не приводящие к увольнению служащих.

Модификация кортежей. Чтобы изменить разряд служащего, мы будем вынуждены модифицировать все кортежи с соответствующим значением атрибута СЛУ_НОМ. В противном случае будет нарушена естественная FD СЛУ_НОМ→СЛУ_УРОВ (у одного служащего имеется только один разряд).

Возможная декомпозиция

Для преодоления этих трудностей можно произвести декомпозицию переменной отношения СЛУЖАЩИЕ_ПРОЕКТЫ_ЗАДАНИЯ на две переменных отношений – СЛУЖ {СЛУ_НОМ, СЛУ_УРОВ, СЛУ_ЗАРП} и СЛУЖ_ПРО_ЗАДАН {СЛУ_НОМ, ПРО_НОМ, СЛУ_ЗАДАН}. На основании теоремы Хита эта декомпозиция является декомпозицией без потерь, поскольку в исходном отношении имелась FD {СЛУ_НОМ, ПРО_НОМ}→СЛУ_ЗАДАН. На рис 11 показаны диаграммы множеств FD этих отношений, а на рис 12– их значения.

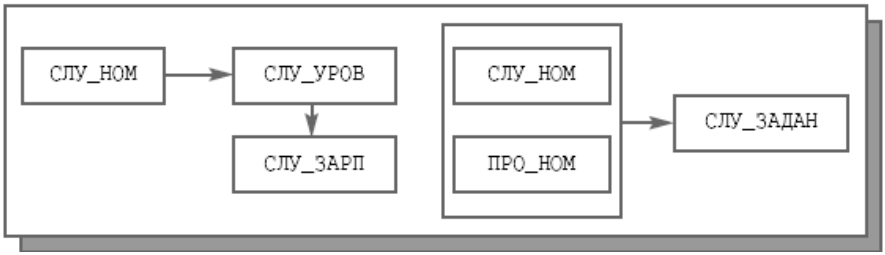

Рисунок 11. Диаграммы FD в переменных отношений СЛУЖ и СЛУЖ_ПРО_ЗАДАН

Теперь мы можем легко справиться с операциями обновления.

Добавление кортежей. Чтобы сохранить данные о принятом на работу служащем, который еще не участвует ни в каком проекте, достаточно добавить соответствующий кортеж в отношение СЛУЖ.

Удаление кортежей. Если кто-то из служащих прекращает работу над проектом, достаточно удалить соответствующий кортеж из отношения СЛУЖ_ПРО_ЗАДАН. При увольнении служащего нужно удалить кортежи с соответствующим значением атрибута СЛУ_НОМ из отношений СЛУЖ и СЛУЖ_ПРО_ЗАДАН.

Модификация кортежей. Если у служащего меняется разряд (и, следовательно, размер зарплаты), достаточно модифицировать один кортеж в отношении СЛУЖ.

Значение переменной отношения СЛУЖ

СЛУ_НОМ	СЛУ_УРОВ	СЛУ_ЗАРП
2934	2	22400.00
2935	3	29600.00
2936	1	20000.00
2937	1	20000.00

Значение переменной отношения СЛУЖ_ПРО_ЗАДАН

СЛУ_НОМ	ПРО_НОМ	СЛУ_ЗАДАН
2934	1	A
2935	1	B
2936	1	C
2937	1	D
2934	2	D
2935	2	C
2936	2	B
937	2	A

Рисунок 12. Значения переменных отношений

Вторая нормальная форма

Как видно, на рисунке отсутствуют FD, не являющиеся минимальными. Наличие таких FD вызывало аномалии обновления. Проблема заключалась в том, что атрибут СЛУЖ_УРОВ относился к сущности служащий, в то время как первичный ключ идентифицировал сущность задание_служащего_в_проекте.

Переменная отношения находится во второй нормальной форме (2NF) тогда и только тогда, когда она находится в первой нормальной форме, и каждый неключевой атрибут32) минимально функционально зависит от первичного ключа33).

Переменные отношений СЛУЖ и СЛУЖ_ПРО_ЗАДАН находятся в 2NF (все неключевые атрибуты отношений минимально зависят от первичных ключей СЛУ_НОМ и

{СЛУ_НОМ, ПРО_НОМ} соответственно). Переменная отношения СЛУЖАЩИЕ_ПРОЕКТЫ_ЗАДАНИЯ не находится в 2NF (например, FD {СЛУ_НОМ, ПРО_НОМ}→СЛУ_УРОВ не является минимальной). Любая переменная отношения, находящаяся в 1NF, но не находящаяся в 2NF, может быть приведена к набору переменных отношений, находящихся в 2NF. В результате декомпозиции мы получаем набор проекций исходной переменной отношения, естественное соединение значений которых воспроизводит значение исходной переменной отношения (т. е. это декомпозиция без потерь). Для переменных отношений СЛУЖ и СЛУЖ_ПРО_ЗАДАН исходное отношение СЛУЖАЩИЕ_ПРОЕКТЫ_ЗАДАНИЯ воспроизводится их естественным соединением по общему атрибуту СЛУ_НОМ.

Заметим, что допустимое значение переменной отношения СЛУЖ может содержать кортежи, информационное наполнение которых выходит за пределы допустимых значений переменной отношения СЛУЖАЩИЕ_ПРОЕКТЫ_ЗАДАНИЯ. Например, в теле отношения СЛУЖ может находиться кортеж с данными о служащем с номером 2938, который еще не участвует ни в одном проекте. Наличие такого кортежа не влияет на результат естественного соединения, тело которого все равно будет совпадать с телом допустимого значения переменной отношения СЛУЖАЩИЕ_ПРОЕКТЫ_ЗАДАНИЯ.

6.2. Нетранзитивные функциональные зависимости и третья нормальная форма

В произведенной декомпозиции переменной отношения СЛУЖАЩИЕ_ПРОЕКТЫ_ЗАДАНИЯ множество FD переменной отношения СЛУЖ_ПРО_ЗАДАН предельно просто – в единственной нетривиальной функциональной зависимости детерминантом является возможный ключ. При использовании этой переменной отношения какие-либо аномалии обновления не возникают. Однако переменная отношения СЛУЖ не является такой же совершенной.

Аномалии обновлений, возникающие из-за наличия транзитивных функциональных зависимостей

Функциональные зависимости переменной отношения СЛУЖ по-прежнему порождают некоторые аномалии обновления. Они вызываются наличием транзитивной FD СЛУ_НОМ⟶СЛУ_ЗАРП (через FD СЛУ_НОМ⟶СЛУ_УРОВ и СЛУ_УРОВ⟶СЛУ_ЗАРП). Эти аномалии связаны с избыточностью хранения значения атрибута СЛУ_ЗАРП в каждом кортеже, характеризующем служащих с одним и тем же разрядом.

Добавление кортежей. Невозможно сохранить данные о новом разряде (и соответствующем ему размере зарплаты), пока не появится служащий с новым разрядом. (Первичный ключ не может содержать неопределенные значения.)

Удаление кортежей. При увольнении последнего служащего с данным разрядом мы утратим информацию о наличии такого разряда и соответствующем размере зарплаты.

Модификация кортежей. При изменении размера зарплаты, соответствующей некоторому разряду, мы будем вынуждены изменить значение атрибута СЛУ_ЗАРП в кортежах всех служащих, которым назначен этот разряд (иначе не будет выполняться FD СЛУ_УРОВ⟶СЛУ_ЗАРП).

Возможная декомпозиция

Для преодоления этих трудностей произведем декомпозицию переменной отношения СЛУЖ на две переменных отношений – СЛУЖ1 {СЛУ_НОМ, СЛУ_УРОВ} и УРОВ {СЛУ_УРОВ, СЛУ_ЗАРП}. По теореме Хита, это снова декомпозиция без потерь по причине наличия, например, FD СЛУ_НОМ⟶СЛУ_УРОВ.

Как видно, это преобразование обратимо, т. е. любое допустимое значение исходной переменной отношения СЛУЖ является естественным соединением значений отношений СЛУЖ1 и УРОВ. Также можно заметить, что мы избавились от трудностей при выполнении операций обновления.

Добавление кортежей. Чтобы сохранить данные о новом разряде, достаточно добавить соответствующий кортеж к отношению УРОВ.

Удаление кортежей. При увольнении последнего служащего, обладающего данным разрядом, удаляется соответствующий кортеж из отношения СЛУЖ1, и данные о разряде сохраняются в отношении УРОВ.

Модификация кортежей. При изменении размера зарплаты, соответствующей некоторому разряду, изменяется значение атрибута СЛУ_ЗАРП ровно в одном кортеже отношения УРОВ.

Третья нормальная форма

Трудности, которые мы испытывали, были связаны с наличием транзитивной FD СЛУ_НОМ→СЛУ_ЗАРП. Наличие этой FD на самом деле означало, что атрибут СЛУ_ЗАРП характеризовал не сущность служащий, а сущность разряд.

Переменная отношения находится в третьей нормальной форме (3NF) в том и только в том случае, когда она находится во второй нормальной форме, и каждый неключевой атрибут нетранзитивно функционально зависит от первичного ключа.

Значение переменной отношения **СЛУЖ1**

СЛУ_НОМ	СЛУ_УРОВ
2934	2
2935	3
2936	1
2937	1

Значение переменной отношения **УРОВ**

СЛУ_УРОВ	СЛУ_ЗАРП
2	22400.00
3	29600.00
1	20000.00

Рисунок 13 Тела отношений СЛУЖ1 и УРОВ

Отношения СЛУЖ1 и УРОВ оба находятся в 3NF (все неключевые атрибуты нетранзитивно зависят от первичных ключей СЛУ_НОМ и СЛУ_УРОВ). Отношение СЛУЖ не находится в 3NF (FD СЛУ_НОМ→СЛУ_ЗАРП является транзитивной). Любое отношение, находящееся в 2NF, но не находящееся в 3NF, может быть приведено к набору отношений, находящихся в 3NF. Мы получаем набор проекций исходного отношения, естественное соединение которых воспроизводит исходное отношение (т. е. это декомпозиция без потерь). Для отношений СЛУЖ1 и УРОВ исходное отношение СЛУЖ воспроизводится их естественным соединением по общему атрибуту СЛУ_УРОВ.

Заметим, что допустимые значения отношения УРОВ могут содержать кортежи, информационное наполнение которых выходит за пределы тела отношения СЛУЖ. Например, в теле отношения УРОВ может находиться кортеж с данными о разряде 4, который еще не присвоен ни одному служащему. Наличие такого кортежа не влияет на результат естественного соединения, который все равно будет являться допустимым значением отношения СЛУЖ.

Независимые проекции отношений. Теорема Риссанена

Обратите внимание, что для переменной отношения СЛУЖ {СЛУ_НОМ, СЛУ_УРОВ, СЛУ_ЗАРП}, кроме декомпозиции на отношения СЛУЖ1 {СЛУ_НОМ, СЛУ_УРОВ} и УРОВ {СЛУ_УРОВ, СЛУ_ЗАРП}, возможна и декомпозиция на отношения СЛУЖ1 {СЛУ_НОМ, СЛУ_УРОВ} и СЛУЖ_ЗАРП {СЛУ_НОМ, СЛУ_ЗАРП}36). Оба отношения, полученные путем второй декомпозиции, находятся в 3NF, и эта декомпозиция также является декомпозицией без потерь. Тем не менее вторая декомпозиция, в отличие от первой, не устраняет проблемы, связанные с обновлением отношения СЛУЖ. Например, по-прежнему невозможно сохранить данные о разряде, которым не обладает ни один служащий. Посмотрим, с чем это связано.

Отношения СЛУЖ1 и УРОВ могут обновляться независимо (являются независимыми проекциями), и при этом

результат их естественного соединения всегда будет таким, как если бы обновлялось исходное отношение СЛУЖ. Это происходит потому, что FD отношения СЛУЖ трансформировались в индивидуальные ограничения первичного ключа отношений СЛУЖ1 и УРОВ. При второй декомпозиции FD СЛУ_УРОВ⟶СЛУ_ЗАРП трансформируется в ограничение целостности сразу для двух отношений (такого рода ограничения целостности называются ограничениями базы данных, и их поддержка гораздо более накладна с технической точки зрения). Понятно, что в процессе нормализации декомпозиция отношения на независимые проекции является предпочтительной. Необходимые и достаточные условия независимости проекций отношения обеспечивает теорема Риссанена.

Теорема Риссанена

Проекции r1 и r2 отношения r являются независимыми тогда и только тогда, когда:

- каждая FD в отношении r логически следует37) из FD в r1 и r2;
- общие атрибуты r1 и r2 образуют возможный ключ хотя бы для одного из этих отношений.

Мы не будем приводить доказательство этой теоремы, но продемонстрируем ее верность на примере двух показанных выше декомпозиций отношения СЛУЖ. В первой декомпозиции (на проекции СЛУЖ1 и УРОВ) общий атрибут СЛУ_УРОВ является возможным (и первичным) ключом отношения УРОВ, а единственная дополнительная FD отношения СЛУЖ (СЛУ_НОМ⟶СЛУ_ЗАРП) логически следует из FD СЛУ_НОМ⟶СЛУ_УРОВ и СЛУ_УРОВ⟶СЛУ_ЗАРП, выполняемых для отношений СЛУЖ1 и УРОВ соответственно. Вторая декомпозиция удовлетворяет второму условию теоремы Риссанена (СЛУ_НОМ является первичным ключом в каждом из отношений СЛУЖ1 и СЛУ_ЗАРП), но FD СЛУ_УРОВ⟶ СЛУ_ЗАРП не выводится из FD СЛУ_НОМ⟶СЛУ_УРОВ и СЛУ_НОМ⟶СЛУ_ЗАРП.

Атомарным отношением называется отношение, которое невозможно декомпозировать на независимые проекции. Далеко не всегда для неатомарных (не являющихся атомарными) отношений требуется декомпозиция на атомарные проекции. Например, отношение СЛУЖ2 {СЛУ_НОМ, СЛУ_ЗАРП, ПРО_НОМ} с множеством FD {СЛУ_НОМ→СЛУ_ЗАРП, СЛУ_НОМ→ПРО_НОМ} не является атомарным (возможна декомпозиция на независимые проекции СЛУЖ3 {СЛУ_НОМ, СЛУ_ЗАРП} и СЛУЖ4 {СЛУ_НОМ, ПРО_НОМ}). Но эта декомпозиция не улучшает свойства отношения СЛУЖ2 и поэтому не является осмысленной. Другими словами, при выборе способа декомпозиции нужно стремиться к получению независимых проекций, но не обязательно атомарных.

Лекция № 7 «Двухуровневые модели архитектуры клиент сервер»

Двухуровневые модели архитектуры клиент-сервер фактически являются результатом распределения пяти указанных ранее групп функций стандартного интерактивного приложения между двумя процессами, выполняемыми на двух платформах: компьютере клиента и на сервере.

7.1 Модель удаленного управления данными (модель файлового сервера)

Увеличение сложности задач, появление персональных компьютеров и локальных вычислительных сетей явились предпосылками появления новой архитектуры «файл-сервер». Эта архитектура баз данных с сетевым доступом предполагает назначение одного из компьютеров сети в качестве выделенного сервера, на котором будут храниться файлы базы данных.

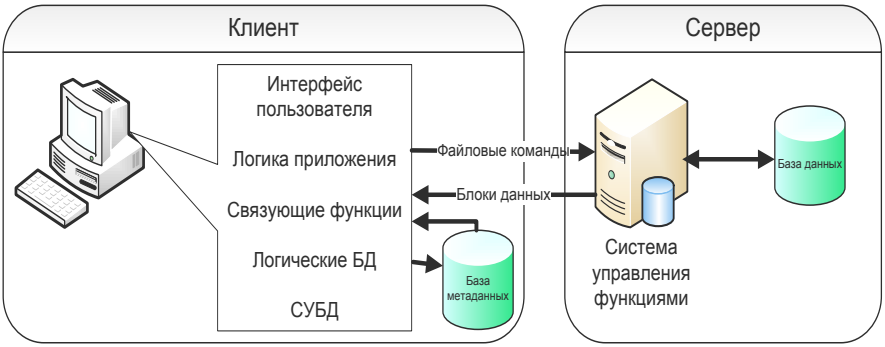

Рисунок 14. Модель файлового сервера

В модели удаленного управления данными или модели файлового сервера (File Server, FS) презентационная логика и бизнес-логика располагаются на клиентской части. В модели файл сервер файлы базы данных хранятся на сервере, клиент обращается к серверу с файловыми командами, а механизм управления всеми информационными ресурсами находится на компьютере клиента. Распределение компонентов приложения в модели файл-сервер представлено на рисунке 14.

В соответствии с запросами пользователей файлы с файл-сервера передаются на рабочие станции пользователей, где и осуществляется основная часть обработки данных. Центральный сервер выполняет в основном только роль хранилища файлов, не участвуя в обработке самих данных.

Достоинство данной модели состоит в том, что приложение разделено на два взаимодействующих процесса. При этом сервер (серверный процесс) может обслуживать множество клиентов, которые обращаются к нему с запросами. Собственно СУБД должна находиться в этой модели на компьютере клиента.

Работа построена следующим образом:

База данных в виде набора файлов находится на жестком диске специально выделенного компьютера (файлового сервера);

Существует локальная сеть, состоящая из клиентских компьютеров, на каждом из которых установлены СУБД и приложение для работы с БД;

На каждом из клиентских компьютеров пользователи имеют возможность запустить приложение. Используя предоставляемый приложением пользовательский интерфейс, он инициирует обращение к БД на выборку/обновление информации;

Все обращения к БД идут через СУБД, которая инкапсулирует внутри себя все сведения о физической структуре БД, расположенной на файловом сервере;

СУБД инициирует обращения к данным, находящимся на файловом сервере, в результате которых часть файлов БД копируется на клиентский компьютер и обрабатывается, что обеспечивает выполнение запросов пользователя (осуществляются необходимые операции над данными);

При необходимости (в случае изменения данных) данные отправляются назад на файловый сервер с целью обновления БД;

Результат СУБД возвращает в приложение;

Приложение, используя пользовательский интерфейс, отображает результат выполнения запросов.

В рамках архитектуры «файл-сервер» были выполнены первые версии популярных так называемых настольных СУБД, таких, как dBase, ранние версии FoxPro и пр.

При этом алгоритм выполнения клиентского запроса сводится к следующему:

1) Запрос формулируется в командах языка манипулирования данными (ЯМД).

2) СУБД переводит этот запрос в последовательность файловых команд.

3) Каждая файловая команда вызывает перекачку блока информации на компьютер клиента, а СУБД анализирует полученную информацию, и если в полученном блоке не содержится ответ на запрос, принимается решение о перекачке следующего блока информации и т.д.

4) Перекачка информации с сервера на клиентский компьютер производится до тех пор, пока не будет получен ответ на запрос клиента.

Рассмотренная модель имеет следующие недостатки:

- высокий сетевой трафик, который связан с передачей по сети множества блоков и файлов, необходимых приложению;
- узкий спектр операций манипулирования с данными, определяемый только файловыми командами;
- отсутствие адекватных средств безопасности доступа к данным (защита только на уровне файловой системы);

Недостаточно развитый аппарат транзакций служит потенциальным источником ошибок в плане нарушения смысловой и ссылочной целостности информации при одновременном внесении изменений в одну и ту же запись

При выполнении сложного запроса (например, сборке аналитического отчета) значительная часть базы данных прокачивается по сети на рабочую станцию клиента и там обрабатывается процессором рабочей станции. Быстродействие такой системы зависит от быстродействия диска сервера, скорости передачи данных по сети, мощности процессора рабочей станции, объема ее ОЗУ и некоторых других факторов. Центральный процессор сервера играет второстепенную роль и должен просто обеспечивать передачу потока данных с сетевого канала на диск и обратно, по возможности не внося замедления в этот процесс. Главным в таком подходе является то, что иногда достаточно большие части базы данных передаются по сети на рабочую станцию для последующей обработки. Если несколько станций одновременно выполняют сборку отчетов, то всем им передается некоторая часть базы данных и естественно система "тормозит". Когда выполняются менее накладные операции, типа ввода нового документа, то объем передачи данных значительно меньше, правда в реальности ввод документа, как правило, сопровождается поиском данных в справочных таблицах, вычислением параметров документа и т.п., что также может порождать передачу большого количества

информации с диска сервера на рабочую станцию. Не следует также забывать о необходимости синхронизации доступа рабочих станций к данным. Поскольку вся обработка ведется на уровне рабочих станций, а файл-сервер просто играет роль разделяемого дискового устройства, задачи синхронизации решаются в таких системах с помощью организации различных файлов блокировок на диске файл-сервера. В них каждая рабочая станция записывает информацию о данных, которые она модифицирует в данный момент, а при попытке считать данные проверяет, не заняты ли эти данные другой рабочей станцией. Несмотря на ряд недостатков такие, как "висячие" блокировки при аварийном выключении рабочих станций, "торможение" всей системы при модификации большого числа записей и т.д., способ вполне работоспособен.

Скорость работы информационной системы в архитектуре файл-сервер напрямую связана с объемом обрабатываемой базы данных. Скорость работы начинает существенно уменьшаться, когда база данных достигает объема свыше 200-300Мб и при приближении к 1Гб практически просто перестает работать. Цифры конечно приблизительные и зависят от используемого программного обеспечения и формата базы данных. Например, при использовании формата таблиц базы данных Paradox торможение наступает значительно позже, чем при использовании формата DBase. В этом случае конечные пользователи и администраторы баз данных идут на различные хитрости: регулярно закрывают старую базу и открывают новую, пытаются удалить старые данные и т.п. Однако во многих случаях данные нужны за длительный промежуток времени и предпочтительно в динамике, а не в виде отдельных кусков. Временным решением проблемы в такой ситуации может быть увеличение пропускной способности сети.

В файл-серверных информационных системах механизм транзакций представляет собой ни что иное, как просто блокировку всей базы данных до завершения выполнения критических по времени операций одной из рабочих станций. Откат возможен только при сохранении работоспособности

рабочей станции, инициировавшей транзакцию. Тем самым, надежность таких систем невысока. При этом поддержка целостности данных возлагается на клиентские приложения, что приводит к их усложнению. Однако файл-серверные базы данных привлекают своей простотой, удобством использования и доступностью. Поэтому файл-серверные информационные системы до сих пор представляют интерес для малых рабочих групп и, более того, нередко используются в качестве информационных систем масштаба предприятия.

7.2 Модель удаленного доступа к данным

В модели удаленного доступа к данным (Remote Data Access – RDA) база данных хранится на сервере. Там же находится и ядро СУБД. На компьютере клиента располагаются презентационная логика и бизнес-логика приложения. Клиент обращается к серверу с запросами на языке SQL. Структура модели удаленного доступа к данным приведена на рисунке 15.

Система управления базами данных в этом случае реализована в виде SQL-сервера – специальной программы, управляющей удаленной базой данных. SQL-сервер обеспечивает интерпретацию запроса, его выполнение в базе данных, формирование результата выполнения запроса и выдачу его приложению-клиенту. При этом ресурсы клиентского компьютера не участвуют в физическом выполнении запроса; клиентский компьютер лишь отсылает запрос к серверной БД и получает результат, после чего интерпретирует его необходимым образом и представляет пользователю. Так как клиентскому приложению посылается результат выполнения запроса, по сети «путешествуют» только те данные, которые необходимы клиенту. В итоге снижается нагрузка на сеть. Поскольку выполнение запроса происходит там же, где хранятся данные (на сервере), нет необходимости в пересылке больших пакетов данных. Кроме того, SQL-сервер, если это возможно, оптимизирует полученный запрос таким образом, чтобы он был выполнен в минимальное время с наименьшими накладными расходами.

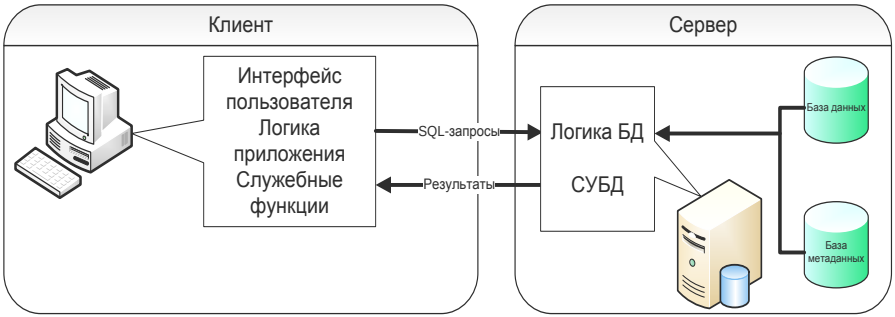

Рисунок 15. Модель удаленного доступа к данным

Все это повышает быстродействие системы и снижает время ожидания результата запроса. При выполнении запросов сервером существенно повышается степень безопасности данных, поскольку правила целостности данных определяются в базе данных на сервере и являются едиными для всех приложений, использующих эту БД. Таким образом, исключается возможность определения противоречивых правил поддержания целостности. Мощный аппарат транзакций, поддерживаемый SQL-серверами, позволяет исключить одновременное изменение одних и тех же данных различными пользователями и предоставляет возможность откатов к первоначальным значениям при внесении в БД изменений, закончившихся аварийно.

Работа клиент-серверного приложения, реализованного в модели удаленного доступа к данным, построена следующим образом:

1) База данных в виде набора файлов находится на жестком диске специально выделенного компьютера (сервера сети);

2) СУБД располагается также на сервере сети;

3) Существует локальная сеть, состоящая из клиентских компьютеров, на каждом из которых установлено клиентское приложение для работы с БД;

4) На каждом из клиентских компьютеров пользователи имеют возможность запустить приложение. Используя предоставляемый приложением пользовательский интерфейс, он инициирует обращение к СУБД, расположенной на сервере, на выборку/обновление информации. Для общения используется специальный язык запросов SQL, т.е. по сети от клиента к серверу передается лишь текст запроса;

5) СУБД инкапсулирует внутри себя все сведения о физической структуре БД, расположенной на сервере;

6) СУБД инициирует обращения к данным, находящимся на сервере, в результате которых на сервере осуществляется вся обработка данных и лишь результат выполнения запроса копируется на клиентский компьютер. Таким образом, СУБД возвращает результат в приложение;

7) Приложение, используя пользовательский интерфейс, отображает результат выполнения запросов.

Поскольку СУБД в данном случае только получает и интерпретирует запросы от клиентских приложений, возвращая результаты их выполнения, такая модель часто называют моделью с пассивным сервером базы данных.

Рассмотрим, как выглядит разграничение функций между сервером и клиентом. Приложение-клиент реализует следующие функции:

Посылка запросов серверу;

Интерпретация результатов запросов, полученных от сервера;

Бизнес-логика приложения;

Представление результатов пользователю в некоторой форме (интерфейс пользователя).

Серверная часть реализует следующие функции:

Прием запросов от приложений-клиентов;

Интерпретация запросов;

Оптимизация и выполнение запросов к БД;

Отправка результатов приложению-клиенту;

Обеспечение системы безопасности и разграничение доступа;

Управление целостностью БД.

Реализация стабильности многопользовательского режима работы.

Как правило, SQL-сервер обслуживается отдельным сотрудником или группой сотрудников (администраторы SQL-сервера). Они управляют физическими характеристиками баз данных, производят оптимизацию, настройку и переопределение различных компонентов БД, создают новые БД, изменяют существующие и т.д., а также выдают привилегии (разрешения на доступ определенного уровня к конкретным БД, SQL-серверу) различным пользователям.

Преимущества данной модели по отношению к модели файл-сервер состоят в следующем:

- перенос компонента представления и прикладного компонента на клиентский компьютер существенно разгружает сервер БД, сводя к минимуму общее число выполняемых процессов в операционной системе;
- сервер БД освобождается от несвойственных ему функций, а процессор или процессоры сервера целиком загружаются операциями обработки данных запросов и транзакций;
- резко уменьшается загрузка сети, так как по ней от клиентов к серверу передаются не запросы на ввод-вывод в файловой терминологии, а запросы на языке SQL, объем которых существенно меньше. В ответ же на эти запросы клиент получает только данные, соответствующие запросу, а не блоки файлов.

Основным достоинством модели RDA является унификация интерфейса клиент-сервер, т.е. стандартным при общении приложения клиента и сервера становится язык SQL.

Недостатки данной модели:

- запросы на языке SQL при интенсивной работе клиентской части приложения могут существенно загрузить сеть;
- так как в этой модели на компьютере клиента располагаются и презентационная логика, и бизнес-логика

приложения, при повторении аналогичных функций в других приложениях код, соответствующей бизнес-логики, должен быть повторен для каждого клиентского приложения, что вызывает излишнее их дублирование;

- так как сервер в этой модели играет пассивную роль, функции управления информационными ресурсами должны выполняться на компьютере клиента.

Также как и в модели файлового сервера, объем данных передаваемых по сети может быть существенным, что увеличивает трафик и снижает скорость выполнения прикладных задач, связанных с обработкой больших объемов данных. Но задачи безопасности и поддержки целостности в данной модели решаются централизованно сервером СУБД, что, несомненно, повышает безопасность и надежность систем, построенных на основе модели удаленного доступа к данным. Информационные системы, построенные на основе архитектуры удаленного доступа, активно используются для автоматизации деятельности небольших рабочих групп и используются в качестве информационных систем масштаба предприятия. Характерным примером системы управления базами данных для построения таких систем является СУБД Microsoft Access, которая входит в состав профессионального выпуска Microsoft Office (что позволяет сэкономить на лицензии), имеет богатый набор языковых и диалоговых средств, удобна в использовании, как профессионалами, так и начинающими разработчиками.

7.3 Модель сервера базы данным

Для исключения недостатков модели удаленного доступа к данным необходимо выполнение следующих условий:

1) БД в каждый момент времени должна отражать текущее состояние предметной области, которое определяется не только собственно данными, но и связями между объектами данных, т.е. данные, которые хранятся в БД, в каждый момент времени должны быть непротиворечивыми.

2) БД должна отражать некоторые правила предметной области и законы, по которым она функционирует, или бизнес-правила (Business Rules). Например, завод может нормально

работать, только если на складе имеется достаточный (страховой) запас деталей определенной номенклатуры, а деталь может быть запущена в производство, только если на складе имеется достаточно материала для ее изготовления, и т.д.

3) Обеспечение постоянного контроля за состоянием БД, отслеживание всех изменений и адекватная реакция на них. Например, при достижении некоторым измеряемым параметром критического значения должно произойти отключение определенной аппаратуры, при уменьшении товарного запаса ниже допустимой нормы должна быть сформирована заявка конкретному поставщику на поставку соответствующего товара и т. п.

4) Возникновение некоторой ситуации в БД должно четко и оперативно влиять на ход выполнения прикладной задачи.

5) Совершенствование контроля типов данных СУБД. В настоящее время СУБД контролирует синтаксически только стандартно-допустимые типы данных, т. е. которые определены в языке описания данных (Data Definition Language, DDL) –, являющемся частью SQL. Однако в реальных предметных областях существуют данные, которые несут в себе еще и семантическую составляющую, например координаты объектов или единицы измерений.

Основу модели сервера базы данных составляют:

механизм хранимых процедур, как средство программирования SQL-сервера;

механизм триггеров как механизм отслеживания текущего состояния информационного хранилища;

механизм ограничений на пользовательские типы данных, который иногда называют механизмом поддержки доменной структуры.

Модель сервера базы данных представлена на рисунке 16. В этой модели бизнес-логика разделена между клиентом и сервером. На сервере бизнес-логика реализована в виде хранимых процедур – специальных программных модулей, которые хранятся в БД и управляются непосредственно СУБД.

Клиентское приложение обращается к серверу с командой запуска хранимой процедуры, а сервер выполняет эту процедуру и регистрирует все предусмотренные изменения в БД. Сервер возвращает клиенту данные выполненного запроса, которые требуются клиенту либо для вывода на экран, либо для выполнения части бизнес-логики. При этом трафик обмена информацией между клиентом и сервером резко уменьшается.

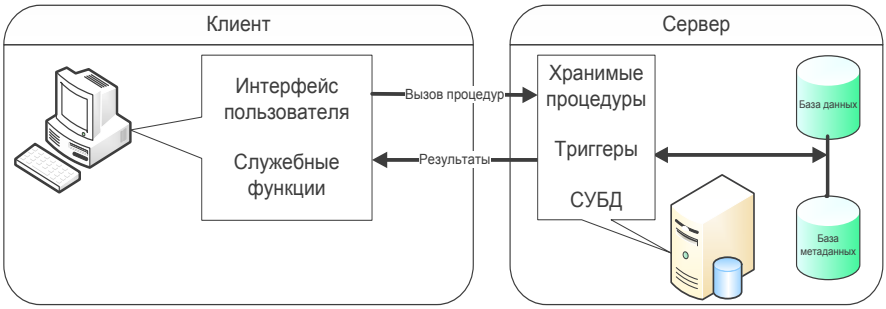

Рисунок 16. Модель сервера базы данных

Централизованный контроль в модели сервера баз данных выполняется с использованием механизма триггеров, которые также являются частью БД. Термин «триггер», взятый из электроники, семантически очень точно характеризует механизм отслеживания специальных событий, связанных с состоянием БД. Триггер является как бы некоторым устройством, который срабатывает при возникновении определенного события в БД. Ядро СУБД проводит мониторинг всех событий, вызывающих созданные и описанные триггеры в БД, и при возникновении такого события сервер запускает соответствующий триггер. Каждый триггер представляет собой также некоторую программу, которая выполняется с базой данных. С помощью триггеров можно вызывать хранимые процедуры. Механизм использования триггеров предполагает, что при срабатывании одного из них могут возникнуть события, которые вызовут срабатывание других. В данной модели сервер является активным, так как в ней не только клиент, но и сам

сервер, используя механизм триггеров, может быть инициатором обработки данных в БД. Хранимые процедуры и триггеры хранятся в словаре БД и, следовательно, могут быть использованы несколькими клиентами, что существенно уменьшает дублирование алгоритмов обработки данных в разных клиентских приложениях.

В связи с тем, что сервер базы данных в указанной модели хранит в базе данных и выполняет приложения хранимых процедур и триггеров, реализующих бизнес-логику на стороне серверной части, данная модель клиент-серверной архитектуры часто называют моделью с активным сервером базы данных.

Основным недостатком данной модели является очень большая загрузка сервера, так как он обслуживает множество клиентов и выполняет следующие функции:

- осуществляет мониторинг событий, связанных с выполнением разработанных триггеров;
- обеспечивает автоматическое срабатывание триггеров при возникновении связанных с ними событий;
- обеспечивает исполнение внутренней программы каждого триггера;
- запускает хранимые процедуры по запросам пользователей;
- запускает хранимые процедуры из триггеров;
- возвращает требуемые данные клиенту;
- обеспечивает выполнение всех функций СУБД – доступ к данным, контроль и поддержку целостности данных в БД, контроль доступа, обеспечение корректной параллельной работы всех пользователей с единой БД.

Если перенести на сервер большую часть бизнес-логики приложений, то требования к клиентам в этой модели резко уменьшатся. Иногда такую модель называют моделью с «тонким» клиентом, а рассмотренные ранее модели – моделями с «толстым» клиентом. Таким образом, рассмотренные двухуровневые модели архитектуры клиент-сервер развивались от «толстого» клиента и «тонкого» сервера, к «тонкому»

клиенту и «толстому» серверу. Для разгрузки сервера базы данных была предложена трехуровневая модель архитектуры «клиент-сервер», в которую в качестве промежуточного уровня добавлен сервер приложений.

Модель активного сервера баз данных поддерживают большинство современных «промышленных» СУБД: MS SQL Server, Oracle, Informix, Sybase, DB2, InterBase и ряд других [5]. Промышленными они называются из-за того, что именно СУБД этого класса могут обеспечить работу информационных систем масштаба среднего и крупного предприятия, организации, банка.

Лекция № 8 «Трехуровневые и многоуровневые модели архитектуры клиент-сервер»

Трехуровневая модель, являющаяся расширением двухуровневой модели с введенным дополнительным промежуточным уровнем между клиентом и сервером, была предложена для разгрузки сервера.

Трехуровневая модель клиент-сервер приведена на рисунке 17. Промежуточный уровень может содержать один или несколько серверов приложений. В данной модели компоненты приложения делятся между тремя исполнителями: клиентом, сервером приложений и сервером базы данных.

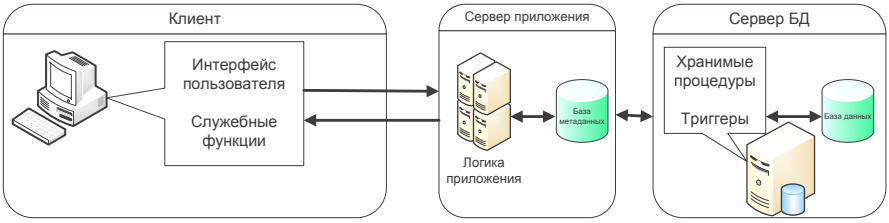

Рисунок 17. Трехуровневая модель клиент-сервер с сервером приложений

Клиент обеспечивает логику представления, включая графический пользовательский интерфейс и локальные редакторы. Клиент может запускать локальный код приложения клиента, который может содержать обращения к локальной БД, расположенной на его компьютере. Клиент исполняет коммуникационные функции приложения, обеспечивающие ему доступ в локальную или глобальную сеть.

Серверы приложений, составляющие новый промежуточный уровень архитектуры модели, спроектированы для исполнения общих не загружаемых функций клиентов. Серверы приложений поддерживают функции клиентов как частей взаимодействующих рабочих групп, сетевую доменную операционную среду и каталоги с данными, а также хранят и исполняют наиболее общие правила бизнес-логики, обеспечивают обмен сообщениями и поддержку запросов.

Серверы баз данных в этой модели занимаются исключительно функциями СУБД: обеспечивают функции создания и ведения БД, поддерживают целостность реляционной БД, обеспечивают функции хранилищ данных. Кроме того, на них возлагаются функции создания резервных копий БД и восстановления БД после сбоев, управления выполнением транзакций и поддержки устаревших (унаследованных) приложений (legacy application).

Что улучшается при использовании трехзвенной архитектуры? Теперь при изменении бизнес-логики более нет необходимости изменять клиентские приложения и обновлять их у всех пользователей. Кроме того, максимально снижаются требования к аппаратуре пользователей.

Работа клиент-серверного приложения, реализованного в модели трехуровневого приложения, построена следующим образом:

1) База данных в виде набора файлов находится на жестком диске специально выделенного компьютера (сервера сети);

2) СУБД располагается также на сервере сети;

3) Существует специально выделенный сервер приложений, на котором располагается программное обеспечение, реализующее общую бизнес-логику приложения;

4) Существует множество клиентских компьютеров, на каждом из которых установлено клиентское приложение, реализующее интерфейс пользователя;

5) На каждом из клиентских компьютеров пользователи имеют возможность запустить приложение–клиент. Используя предоставляемый приложением пользовательский интерфейс, он инициирует обращение к программному обеспечению, расположенному на сервере приложений;

6) Сервер приложений анализирует требования пользователя и формирует запросы к БД. Для общения используется специальный язык запросов SQL, т.е. по сети от сервера приложений к серверу БД передается лишь текст запроса или вызов хранимой процедуры;

7) СУБД инкапсулирует внутри себя все сведения о физической структуре БД, расположенной на сервере.

8) СУБД инициирует обращения к данным, находящимся на сервере, после чего результат выполнения запроса копируется на сервер приложений;

9) Сервер приложений возвращает результат в клиентское приложение (пользователю);

10) Приложение, используя пользовательский интерфейс, отображает результат выполнения запросов.

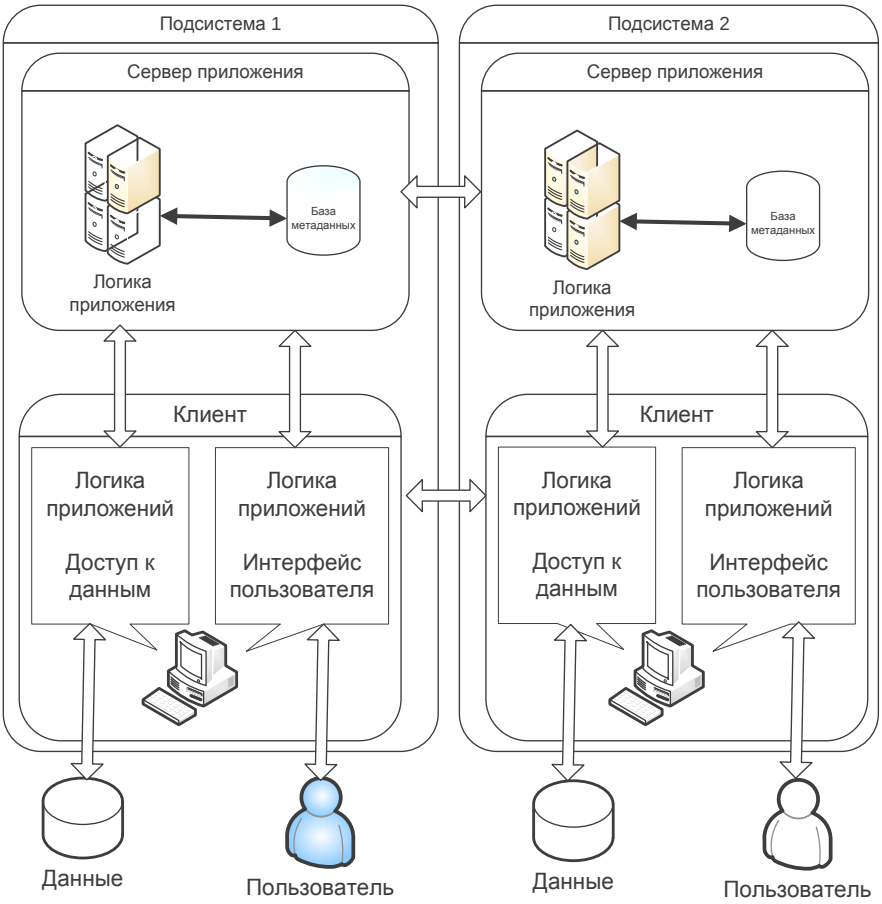

Рисунок 18. Система прямого обмена данными между клиентами и серверами

Данная модель обладает большей гибкостью, чем двухуровневые модели. Наиболее заметны преимущества модели сервера приложений в тех случаях, когда выполняются сложные аналитические расчеты в базе данных, относящихся к области OLAP-приложений.

В этой модели большая часть бизнес-логики клиента изолирована от возможностей встроенного языка SQL,

реализованного в конкретной СУБД, и может быть выполнена на языках программирования Pascal, Visual Basic, C, C++, C# и др., что повышает переносимость системы и ее масштабируемость.

Недостатком модели можно считать более высокие затраты ресурсов компьютеров на обмен информацией между компонентами приложений по сравнению с двухуровневыми моделями.

В последнее время архитектура клиент-сервер получила дальнейшее развитие. Появилась архитектура, называемая равный к равному (peer to peer) (рисунок 6).

Она реализована, например, в СУБД InterBase фирмы Borland и в СУБД Oracle. При этой архитектуре все узлы сети являются равноправными, т.е. они являются и клиентами и серверами одновременно. Из любого узла сети можно запросить и обновить данные и метаданные, хранящиеся на любом другом компьютере сети. При таком подходе деление компьютеров на серверы и клиенты становится условным, однако, поскольку программы-серверы требуют достаточно больших вычислительных ресурсов, архитектура равный-к-равному требует установки в узлах компьютеров с большой оперативной и дисковой памятью. Подобные системы являются в настоящий момент, вероятно, одними из крупнейших существующих распределенных систем, объединяющие миллионы компьютеров.

Лекция № 9 «Сервис-ориентированная архитектура»

В настоящее время наблюдается устойчивый рост интереса к концепции сервис-ориентированной архитектуры (Service-Oriented Architecture – SOA).

SOA – это архитектурная модель для объединения доступных вычислительных ресурсов, таких как: приложения, данные, формы их представления, средства управления и мониторинга, с требованиями достижения желаемых результатов, для потребителей сервисов, которыми могут быть:

конечные пользователи, другие приложения, или другие, подобные, сервисы.

В общем виде SOA предполагает наличие трех основных участников: поставщика сервиса, потребителя сервиса и реестра сервисов (рисунок 19). Взаимодействие участников выглядит достаточно просто: поставщик сервиса регистрирует свои сервисы в реестре, а потребитель обращается к сервису с запросом.

В компании WSO2 разработан реестр/репозиторий для SOA, распространяемый в открытых кодах, — WSO2 Registry. Программное обеспечение имеет Web-интерфейс в стиле wiki для управления метаданными предприятия, снабженный средствами Web 2.0: тэгами, рейтингами и системой комментариев. Поддержку таких средств в WSO2 называют фактором, отличающим реестр/репозиторий от другого открытого проекта в данной области, MuleSource. По словам специалистов WSO2, назначение системы — в хранении с версионным контролем всех метаданных SOA, накапливаемых на предприятии: описаний сервисов, XML-схем, конфигурационной информации. Как указывают в компании, WSO2 Registry — это несложный и высокопроизводительный продукт, обеспечивающий простейшие средства управления жизненным циклом метаданных SOA.

Кроме форматов метаданных реестр поддерживает хранение произвольных файлов, что позволяет создавать каталоги информации предприятия, перечисляющие сервисы, их описания, сведения по служащим и текущим проектам. Имеется возможность использования репозитория в составе Java-приложений — для хранения ресурсов. Функции системы также можно удаленно вызывать по механизму REST с помощью API, выполненного на основе протоколов Atom/AtomPub.

Возможно, что за такими моделями архитектуры информационных систем ближайшее будущее.

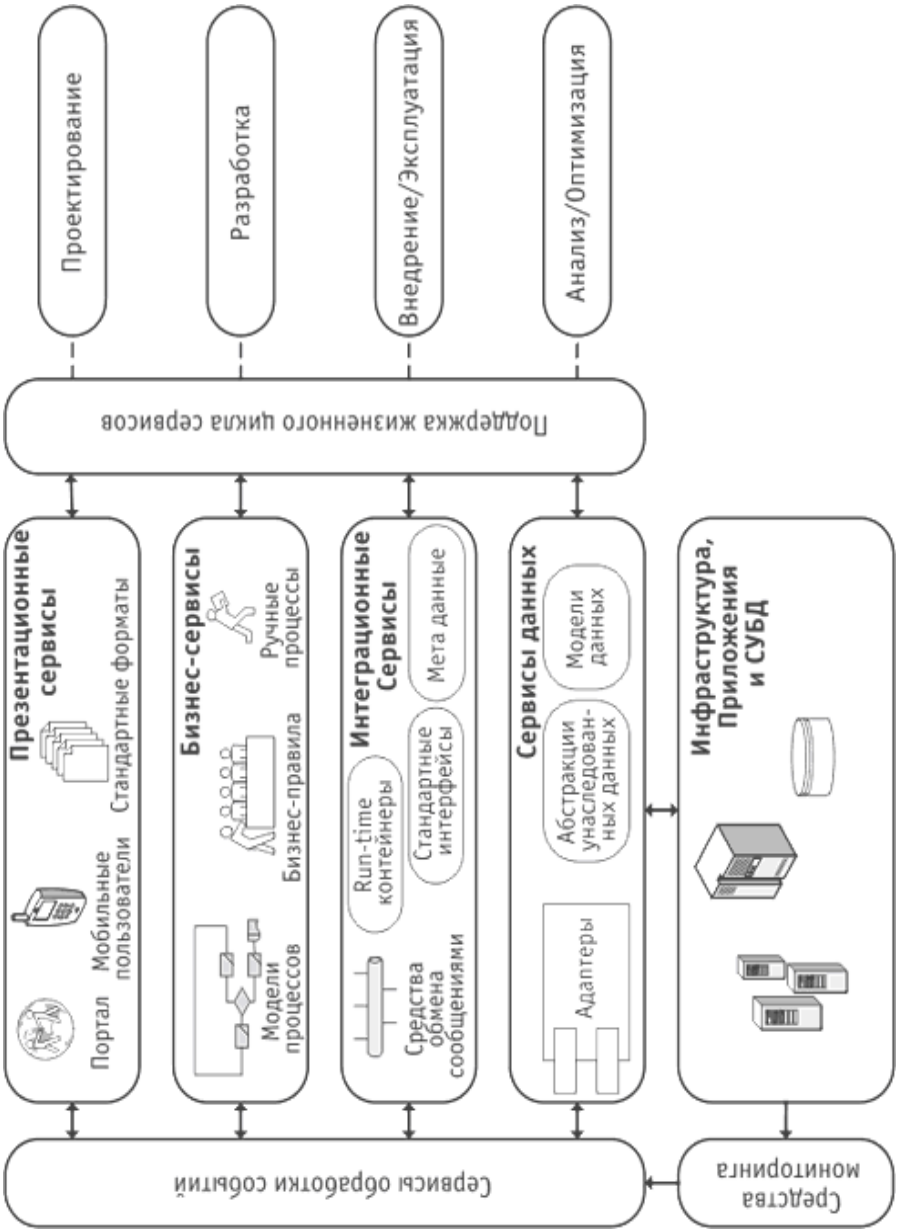

Рисунок 19. Обобщенная схема SOA

Лекция 10 «Краткий обзор систем управления базами данных»

Достаточно часто системы управления базами данных классифицируются на две большие категории: так называемые настольные и серверные СУБД.

10.1 Настольные системы управления базами данных

Настольные СУБД используются для сравнительно небольших задач – небольшой объем обрабатываемых данных, малое количество пользователей. С учетом этого, указанные СУБД имеют относительно упрощенную архитектуру, в частности, функционируют в режиме файл-сервер, поддерживают не все возможные функции СУБД. Например, в таких СУБД может не вестись журнал транзакций, отсутствовать возможность автоматического восстановления базы данных после сбоев и т. п. Тем не менее, такие системы имеют достаточно обширную область применения. Прежде всего, это государственные (муниципальные) учреждения, сфера образования, сфера обслуживания, малый и средний бизнес. Специфика возникающих в таких предметных областях задач заключается в том, что объемы данных не являются катастрофически большими, частота обновлений не бывает слишком высокой, организация территориально обычно расположена в одном небольшом здании, количество пользователей колеблется от одного до 10–15 человек. В подобных условиях использование настольных СУБД для управления информационными системами является вполне оправданным, и они с успехом применяются.

Одними из первых СУБД были так называемые dBase-совместимые программные системы, разработанные разными фирмами. Первой широко распространенной системой такого рода была система dBase III – PLUS (фирма Achton-Tate). Развитый язык программирования, удобный интерфейс, доступный для массового пользователя, способствовали широкому распространению системы. В то же время работа системы в режиме интерпретации обусловливала низкую

производительность на стадии выполнения. Это привело к появлению новых систем-компиляторов, близких к системе dBase III – PLUSA именно, Clipper (фирма Nantucket Inc.), FoxPro (фирма Fox Software), FoxBase+ (фирма Fox Software), Visual FoxPro (фирма Microsoft). Одно время достаточно широко использовалась СУБД PARADOX (фирма Borland International).

В последние годы очень широкое распространение получила система управления базами данных Microsoft Access, которая входит в целый ряд версий пакета Microsoft Office(фирма Microsoft).

10.2 Серверные систему управления базами данных

Серверы баз данных осуществляют целый комплекс действий по управлению данными. Основными их обязанностями являются:

- выполнение пользовательских запросов на выбор и модификацию данных и метаданных, получаемых от клиентских приложений, функционирующих на персональных компьютерах локальной сети;
- хранение и резервное копирование данных;
- поддержка ссылочной целостности данных согласно определенным в базе данных правилам;
- обеспечение авторизованного доступа к данным на основе проверки прав и привилегий пользователей;
- протоколирование операций и ведение журнала транзакций.

Кроме того, современные серверные СУБД обычно предоставляют дополнительный набор сервисов, связанных с обслуживанием хранения и обработки данных, созданием клиентских приложений, сменой СУБД или ее версии, обслуживанием нескольких баз данных, публикацией данных в Internet и т.д. Поскольку в условиях конкуренции между различными производителями СУБД каждый из них стремится к завоеванию как можно большей части рынка, это приводит к тому, что все они пытаются предоставить потенциальному потребителю как можно больше сервисов различного назначения. При выборе СУБД потребитель обычно

ориентируется на то, что поддерживает данная СУБД, с одной стороны, и чем она поддерживается – с другой.

Современных серверы баз данных большинства производителей в той или иной степени обладают следующими характеристиками и возможностями.

Реализация для нескольких платформ. Почти все современные серверы баз данных существуют в нескольких версиях для различных платформ. Исключением из этого правила является Microsoft SQL Server. Однако для данной СУБД это вполне оправданно – компания Microsoft сама производит серверные операционные системы и в отличие от большинства других производителей серверных СУБД может себе позволить создавать серверы баз данных, тесно интегрированные с сервисами операционной системы собственного производства и поддерживающие исключительно их.

Утилиты администрирования. Администрирование сервера баз данных, конечно, – удел профессионалов. Однако и профессионал предпочтет удобные утилиты администрирования унылому окну с интерфейсом командной строки. Наличие удобных утилит администрирования, как ни странно, иногда оказывается одним из решающих факторов при выборе СУБД. Именно поэтому подавляющее большинство современных СУБД обычно поставляется с подобными утилитами.

Резервное копирование данных. Резервное копирование данных и журналов транзакций поддерживается всеми без исключения коммерческими серверными СУБД. Различия между СУБД в поддержке резервного копирования заключаются в том, возможно ли производить резервное копирование в процессе работы пользователей, и если да, то какие пользовательские операции в это время нельзя выполнять. Помимо этого в комплекте поставки некоторых СУБД могут содержаться утилиты для использования различных внешних устройств (например, накопителей на магнитных лентах) в качестве средств хранения резервных копий.

Обслуживание репликаций. Репликация по существу представляет собой гарантированное копирование информации из одной базы в несколько других. Репликации используются для разделения нагрузки между серверами в сети, для перемещения поднаборов данных на вспомогательные серверы, для синхронизации данных на нескольких серверах и многих других целей.

В том или ином виде репликации поддерживаются всеми современными серверными СУБД. Различия могут быть лишь в поддержке тех или иных конкретных сценариев репликаций, например, внесение изменений одновременно на нескольких серверах, возможность шифрования реплицируемых данных и др.

Параллельная обработка данных в многопроцессорных системах. О возможностях повышения производительности с помощью параллельной обработки запросов в многопроцессорных системах начали говорить несколько лет назад, после появления первого продукта такого класса – Oracle Parallel Server. Серверы, поддерживающие параллельную обработку, разрешают нескольким процессорам обращаться к одной базе данных, что позволяет обеспечить высокую скорость обработки транзакций.

В настоящее время, подавляющее большинство производителей современных серверных СУБД поставляют на рынок версии, поддерживающие параллельную обработку данных.

Поддержка OLAP и создания хранилищ данных. OLAP (On-Line Analytical Processing) представляет собой технологию построения многомерных хранилищ данных (Data Warehouses), как правило, агрегатных, то есть являющихся результатом обработки набора данных, нередко состоящего из нескольких таблиц. Такие хранилища данных в последнее время широко используются в системах поддержки принятия решений. Более подробно об идеях, лежащих в основе OLAP, будет рассказано в одной из последующих статей данного цикла, здесь же мы ограничимся лишь упоминанием этой возможности.

Многомерные хранилища данных могут быть реализованы как в виде набора обычных реляционных таблиц, так и в виде нереляционной многомерной базы данных. В последнем случае такое хранилище обычно управляется отдельным сервером. Многие производители серверных СУБД поставляют такие серверы отдельно (Oracle, Informix), некоторые включают их в состав сервера реляционных баз данных (Microsoft SQL Server). Нередко с целью повышения конкурентоспособности подобные OLAP-системы строят многомерные хранилища на основе данных из других СУБД, как это сделано, например, в Microsoft SQL Server OLAP Extensions и в Sybase Adaptive Server IQ.

Распределенные запросы и транзакции. Распределенные транзакции и запросы стали особенно актуальны в последнее время, когда наличие нескольких серверов баз данных в одной организации стало обычным явлением. Нужно отметить, что возможности выполнения распределенного запроса или распределенной транзакции поддерживаются сейчас почти всеми серверными СУБД, по крайней мере, в том случае, когда все вовлеченные в транзакцию серверы – от одного и того же производителя. С этой целью используется механизм двухфазного завершения транзакций, когда на первом этапе серверы, вовлеченные в транзакцию, сигнализируют о готовности ее завершить, а на втором этапе происходит реальная фиксация изменений в базах данных.

Что касается распределенных транзакций с участием сторонних серверов, то они, как правило, реализуются с помощью мониторов транзакций или иных подобных сервисов, например Microsoft Distributed Transaction Coordinator.

Средства проектирования данных. Многие производители серверных СУБД производят также средства анализа бизнес-процессов и проектирования данных, иногда универсальные (как в случае Sybase DataArchitect), а порой ориентированные главным образом на конкретную СУБД (как в случае Oracle Designer/2000). Многие производители СУБД не имеют в своем арсенале собственных средств проектирования

данных, ориентируясь на универсальные CASE-средства. Нередко производители СУБД встраивают в административные утилиты несложные средства проектирования данных, позволяющие визуально редактировать схемы данных, как это сделано, например, в Microsoft SQL Server.

Поддержка собственных и сторонних средств разработки и генераторов отчетов. Многие производители серверных СУБД выпускают также средства разработки и генераторы отчетов. Иногда данные средства разработки используют тот же язык программирования, что применяется при написании триггеров и хранимых процедур (в этом случае, как правило, клиентское приложение должно включать интерпретатор этого языка), что позволяет отлаживать хранимые процедуры, помещая их в клиентское приложение. Типичный пример подобного подхода реализован в Oracle Developer/2000. Однако чаще средства разработки производителей серверных СУБД используют языки программирования, отличные от языков создания серверного кода (характерный пример – четыре средства разработки Microsoft).

Практически все производители серверных СУБД делают все возможное для того, чтобы клиентские приложения для их СУБД можно было создавать с помощью других средств разработки. С этой целью они предоставляют разработчикам описания API клиентской части, ODBC-драйверы, OLE DB-провайдеры, а нередко и объектные модели, позволяя использовать COM-объекты клиентской части в приложениях (как, например, это сделано в клиентских частях Oracle, Microsoft SQL Server, Informix).

Поддержка доступа к данным с помощью Internet. Без поддержки публикации данных в Internet или получения данных от удаленных Internet-клиентов сегодня не обходится практически ни одна коммерческая СУБД, в том числе настольные базы данных. Тем или иным способом производители серверных СУБД поддерживают Web-технологии. Чаще всего эта поддержка осуществляется с помощью Web-серверов собственного производства, либо

посредством создания расширений для существующих Web-серверов, либо просто путем включения в комплект поставки утилит, генерирующих Web-страницы согласно определенному расписанию.

Список литературы

1. Голицына О.Л., Максимов Н.В., Попов И.И. Базы данных: учеб. пособие. - 2-е изд. испр. и доп.- М.:ФОРУМ: ИНФРА-М, 2009.- 400 с.: ил.
2. Гектор Гарсиа-Молина, Джеффри Ульман, Дженифер Уидом. Системы баз данных. Полный курс. Москва, Санкт-Петербург, Киев, Вильямс, 2003
3. Глушаков С.В., Ломотько Д.В. «Базы данных: Учебный курс», М., АСТ, 2000.-504 с., илл.
4. К. Дейт. Введение в системы баз данных. 2-е изд., М.: Наука.1980.
5. К. Дейт. Введение в системы баз данных. 6-е изд., М.; СПб.: Вильямс.- 2000
6. С.Д. Кузнецов. Базы данных: языки и модели. Москва, Бином, 2008
7. Кузнецов С.Д. Основы баз данных: Учебное пособие / С.Д. Кузнецов.- 2-е изд. испр.- М.: Интернет-Университет Информационных технологий; БИНОМ. Лаборатория знаний, 2010.- 484 с.: ил.
8. Хомоненко А.Д., Цыганков В.М., Мальцев М.Г. Базы данных: Учебник для высших учебных заведений/ Под ред. Проф. А.Д. Хомоненко. - 6-е изд.- СПб.: КОРОНА-Век, 2010.- 736 с.