Электронный образовательный ресурс

Национальный Исследовательский Университет МЭИ

Лекции по курсу

Базы данных

Дисциплина базовой части профессионального цикла Б 3.5

Направление подготовки 010400 Прикладная математика и информатика

Бакалаврская программа Математическое и программное обеспечение вычислительных машин и компьютерных сетей

Автор:

Сидорова Н.П.

Москва **2012** **НИУ МЭИ**

Содержание

ЛЕКЦИЯ 1. ОСНОВНЫЕ ПОНЯТИЯ БД

Основные понятия. Роль и место систем управления базами данных (СУБД) в современных автоматизированных системах. Свойства базы данных (БД). Средства и методы анализа предметной области. Функции, структура и архитектура СУБД. Уровни представления данных. Классификация моделей данных, лежащих в основе СУБД.

1.1. Основные понятия БД.

Информационная технология (ИТ) [*information technology* (*IT*)] – общий термин, обозначающий любую технологию, связанную с созданием, хранением, обработкой, использованием, передачей и управлением информацией.

Данные [*data*] (здесь) – любая информация, представленная в форме, пригодной для хранения, передачи и обработки средствами вычислительной техники. Информационные процессы, использующие данные, называются **обработкой данных** [*data processing*] и изучаются *информатикой*.

Банк данных (БнД) – основа современных информационных систем (ИС). БнД - система специальным образом организованных данных, программных, языковых, технических, организационно-методических средств, предназначенных для обеспечения централизованного накопления и коллективного многоцелевого использования данных. БнД создается для поддержки решения различных задач, возникающих в конкретной предметной области.

База данных (БД) [*database* (*DB*)] – структурированная совокупность данных, организованная по единым правилам, включающим принципы описания, хранения и манипулирования этими данными. Как правило, БД является информационной моделью некоторой предметной области. БД можно рассматривать на различных уровнях абстракции, обычно выделяют как минимум два: **физический** (способ представления данных с использованием конкретного аппаратного и программного обеспечения) и **логический** (способ представления данных с точки зрения пользователя).

БД является основой БнД и представляет собой специальным образом организованные данные, хранящиеся во внешней памяти компьютера.

Данные отражают:

- сведения о деятельности (объектах, процессах и т.п.) в конкретной предметной области;
- описания хранимых данных – метаданные – схемы данных (они образуют словарь данных)
- данные, используемые в процессе управления БД (журнал транзакций, контрольная точка и т.д.).

С логической точки зрения БД [1] – абстрактное, самодостаточное логическое определение объектов, операторов и др. элементов, в совокупности составляющих абстрактную машину, с которой взаимодействует пользователь. Она позволяет моделировать структуру данных и их поведение.

Система управления базами данных (СУБД) [database management system (DBMS)] – системное программное обеспечение, служащее для абстракции физического уровня представления данными и управления доступом к данным. Основные функции СУБД: определение данных (то есть работа с метаданными [metadata] – данными, описывающими данные), хранение данных, обработка данных, обеспечение целостности и безопасности данных, импорт/экспорт данных в различных форматах.

Предметная область – часть реального мира, информация из которой используется конечными пользователями БД для решения своих задач. Понятие предметной области БД не имеет точного определения, однако является одним из базовых понятий информатики. Оно предполагает выделение устойчивых связей между именами, понятиями и определенными реалиями внешнего мира, которые не зависят от БД и ее пользователей. Это понятие позволяет ограничить состав данных, которые должны храниться в БД и реализовать работы с данными. В БД нельзя получить ответ на вопрос о данных, которые в ней не хранятся.

1.2. Назначение и свойства БД

Для обработки возрастающего объема информации в настоящее время широко применяют специальные виды информационных технологий – автоматизированные информационные системы (АИС). В самом широком понимании АИС рассматривается как система обработки информации, в которой реализованы функции централизованного хранения и накопления обрабатываемой организованной. Основой любой АИС является БД. Сейчас трудно назвать сферу человеческой деятельности, которая не была бы связана с использованием БД. Органы государственной власти (налоговая инспекция, органы статистики и др.), медицинские учреждения, магазины, банки, предприятия и организации различного уровня используют в процессе своей работы различные информационные системы, и, следовательно, БД. БД позволяют хранить большие объемы информации, оперативно их обрабатывать, делать доступной информацию большому числу людей, которым она нужна для решения разнообразных задач.

Объективные причины внедрения БД:

- Информационные потребности различных пользователей в конкретной предметной области пересекаются;
- Функции создания и ведения информационного фонда и его обработки являются универсальными;
- Современное состояние средств вычислительной техники и программирования.

Различают следующие классы пользователей БД:

- конечный пользователь
- пользователь-программист
- администратор БД.

Конечный пользователь – специалист, использующий данные из БД для решения профессиональных задач (инженер, извлекающий данные об имеющих деталях из БД деталей; менеджер, формирующий заказ на основе БД товаров по заявке клиента; кассир, оформляющий билет на основе БД билетов).

Пользователь-программист – специалист, использующий БД для разработки приложений с БД. Приложение – это программы, использующие данные из БД для решения конкретной задачи.

Администратор БД – специалист, отвечающий за создание и эффективное использование БД. Он обеспечивает качество, сохранность и доступность для использования в служебных целях информации, накопленной в организации. Администратор БД поддерживает эффективное функционирование СУБД и использование компьютерных баз данных, в том числе поддержку схем данных, надежность и отказоустойчивость технических средств всех уровней, доступность и удобство ввода-вывода данных

Функции администратора БД:

- анализ предметной области: описание, выявление ограничений целостности, определение статуса информации, характеристики обработки данных;
- проектирование структуры БД: состав и структура единиц информации и связей между ними, выбор методов упорядочивания данных и доступа к ним, описание структуры средствами языка;
- задание ограничений целостности на основе семантики данных Предметная область;
- защита данных от несанкционированного доступа;
- защита данных от разрушения (резервное копирование);
- восстановление данных;
- анализ эффективности использования;
- развитие БД.

При централизованном управлении на предприятии, использующем БД, есть человек, который несет основную ответственность за данные предприятия. Это – *администратор данных*, который должен разбираться в тех данных, которые используются, и потребностях, которые есть у предприятия них, и понимать нужды предприятия по отношению к данным на уровне высшего управляющего звена в руководстве предприятием. Сам администратор данных также должен относиться к этому звену. В его обязанности входит принятие решений о том, какие данные необходимо вносить в базу данных в первую очередь, а также выработка требований по сопровождению и обработке данных после их занесения в базу данных. В отличие от администратора данных, администратор БД является техническим специалистом, который отвечает за реализацию решений администратора данных. Администратор базы данных должен быть профессиональным специалистом в области информационных технологий. Работа администратора БД заключается в создании самих баз данных и организации технического контроля, необходимого для осуществления решений, принятых администратором данных. Он также несет также ответственность за обеспечение необходимого быстродействия системы и ее техническое обслуживание.

Современные БД обладают следующими свойствами:

- Возможность совместного доступа к данным. Сетевые технологии и клиент-серверные технологии позволяют осуществлять одновременный доступ к данным сразу нескольким сотням пользователей. При этом данные в этом случае физически хранятся на одном устройстве (сервере), работой с которым управляет сервер БД. Совместный доступ к данным означает не только возможность доступа к ним с помощью нескольких существующих приложений базы данных, но и возможность разработки новых приложений для работы с этими же данными. Другими словами, требования новых приложений по доступу к данным могут быть удовлетворены без необходимости добавления новых данных в базу.

- Минимизация избыточности данных связана с тем, что разные приложения используют одни и те же хранимые данные, и, следовательно, нет необходимости дублировать данные для каждого отдельного пользователя или приложения. В системах, не использующих базы данных, каждое приложение имеет свои файлы. Это часто приводит к избыточности хранимых данных и, следовательно, к нерациональному использованию пространства вторичной памяти.

- Непротиворечивость данных позволяет гарантировать, что одни и те же данные, используемые различными пользователями, имеют одинаковые значения. Противоречий можно также избежать, если избыточность не исключается, а контролируется (и это соответствующим образом предусмотрено в СУБД). Тогда СУБД сможет гарантировать, что с точки зрения пользователя база данных никогда не будет противоречивой.

- Поддержка транзакций, которая обеспечивает возможность одновременной работы с данными нескольких пользователей. Транзакция – последовательность операций над БД, отслеживаемая СУБД от ее начала до завершения как единое целое.

- Обеспечение целостности данных. Целостность – одно из важнейших свойств БД, которая позволяет контролировать средствами СУБД семантические соотношения хранящихся в БД данных. Задача обеспечения целостности заключается в гарантированной поддержке правильности данных средствами СУБД.

- Организация защиты данных, которая позволяет ограничить доступ к данным из БД, используя возможности СУБД.

- Увеличение гибкости обслуживания запросов данных связано с наличием развитой системы стандартов при работе с БД.

- Сокращение времени разработки приложений определяется стандартизацией языков работы с СУБД.

- Наличие развитых средств поддержки, которые позволяют решать задачи надежного хранения и восстановления данных после сбоя;

- Поддержка логической и физической независимости программ от данных. Логическая независимость означает возможность изменения логической модели БД без изменения уже существующих программ пользователей. Физическая независимость гарантирует возможность переноса БД на новые устройства хранения данных без изменения имеющихся программ пользователей. Независимость от данных можно определить как невосприимчивость приложений к изменениям в физическом представлении данных и в методах доступа к ним, а это означает, что рассматриваемые приложения не зависят от любых конкретных способов физического представления информации или выбранных методов доступа к ним.

- Стандартизация представления данных в системе. Благодаря централизованному управлению БД администратор БД может обеспечить соблюдение всех необходимых стандартов, регламентирующих представление данных в системе. Стандартизация представления данных наиболее важна с точки зрения обмена данными и их пересылки между системами

1.3. Этапы развития БД

Этапы развития ИТ в области БД связаны с развитием средств вычислительной техники, методов доступа к внешней памяти и развитием работ по разработке моделей БД.

1. В начале 60-х гг. XX века для хранения данных использовались файловые структуры и последовательные накопители данных. Данные хранились в файловых системах, в которых физическая структура хранимых данных жестко которых определялась логической структурой данных. Программы обработки данных в этом случае были жестко привязаны к этой структуре и при ее изменении должны были создаваться заново (зависимость программ от данных – проблема возникает при одновременной работе с этими данными нескольких пользователей со своими программами). Основными проблемами управления такими БД являлись сложности управления избыточностью, разграничения прав доступа и др. Для организации БД использовались большие ЭВМ типа IBM 360/370, ЕС-ЭВМ и мини-ЭВМ типа PDP11. БД Базы данных хранились во внешней памяти центральной ЭВМ. Пользователи БД использовали её в пакетном режиме. Интерактивный режим доступа обеспечивался с помощью консольных терминалов, которые не обладали собственными вычислительными ресурсами.

2. В середине 60-х годов прошлого века корпорация разработала первую СУБД – иерархическую систему IMS (Information Management System). В нашей стране была разработана и использовалась иерархическая СУБД ОКА. Несмотря на то, что IMS является самой первой из всех коммерческих СУБД, она до сих пор остается основной иерархической СУБД, используемой на большинстве крупных мейнфреймов. Фирма General Electric представила свою разработку – систему IDS (Integrated Data Store). На её основе были созданы первые сетевые СУБД. Это оказало большое влияние на информационные системы того поколения. Сетевая СУБД создавалась для представления более сложных взаимосвязей между данными, чем те, которые можно было моделировать с помощью иерархических структур, и послужили основой для разработки первых стандартов БД. Наиболее известной реализацией сетевой СУБД явилась СУБД CODASYL. В нашей стране были разработаны СУБД СЕДАН. В 1965 году на конференции CODASYL (Conference on Data Systems Languages) была сформирована рабочая группа List Processing Task Force, переименованная в 1967 году в группу Data Base Task Group (DBTG). В компетенцию группы DBTG входило определение спецификаций и стандартов для разработки БД и СУБД.

3. 70-80-е гг. XX века характеризовались большим развитием направления БД. В 1970 году Э. Ф. Кодд, работавший в корпорации IBM, опубликовал статью о реляционной модели данных, позволявшей устранить недостатки прежних моделей. В это же время появляются первые экспериментальные реляционные СУБД. Первые коммерческие продукты появились в конце 70-х - начале 80-х годов. Особенно важным в этом отношении является проект System R, разработанный в корпорации IBM в конце 70-х годов (Astrahan et al., 1976). Он продемонстрировал практичность реляционной модели, что достигалось с помощью реализации предусмотренных ею структур данных и требуемых функциональных возможностей. В результате был разработан структурированный язык запросов SQL, который с тех пор стал стандартным языком любых реляционных СУБД. В 80-х годах

были созданы различные коммерческие реляционные СУБД - например, DB2 или SQL/DS корпорации IBM, Oracle корпорации Oracle , др., которые и до настоящего времени являются наиболее развитыми и широко используемыми СУБД.

4. 80- 90 гг. XX века ознаменовались появлением персональных компьютеров и широким распространением персональных БД. Для работы с ними были созданы СУБД FoxPro, Clipper, DbaseIII, Paradox и др. В это же время появляются первые вычислительные сети, для которых разрабатываются серверные СУБД.

5. Современное состояние БД характеризуется применением распределенные сетевые систем, использующих технологию клиент-сервер. Они способные работать на различных платформах. В настоящее время в технологии БД сложилась развитая система стандартов в области языков и протоколов обмена данными, подключение клиентских приложений, средства экспорта данных из различных форматов. Дальнейшее развитие связано с появлением новых типов моделей данных (объектно-ориентированных, многомерных и др.).

1.4. Средства и методы анализа предметной области

БД является информационной моделью предметной области. Понятие *предметной области базы данных* является одним из базовых понятий технологии БД, однако не имеет точного определения. Его использование в контексте изучаемой дисциплины предполагает существование устойчивой во времени соответствия между именами, понятиями и определенными реалиями внешнего мира, не зависящей от самой БД и ее круга пользователей. Таким образом, введение в рассмотрение понятия *предметной области базы данных* позволяет ограничить состав тех данных. Которые будут храниться в БД. *Модель предметной области* отражает наши знания о предметной области. Знания могут быть как в виде неформальных знаний эксперта, так и выражены формально при помощи каких-либо методов и средств. В качестве таких средств могут выступать текстовые описания предметной области, наборы должностных инструкций, правила ведения дел в компании и т.п. Однако опыт показывает, что текстовый способ представления модели предметной области крайне неэффективен, его трудно анализировать и использовать при моделировании структур данных. Более информативными и полезными при разработке баз данных являются описания предметной области, выполненные при помощи специализированных графических моделей, которые строятся на основе анализа предметной области. Существует большое количество методик описания предметной области. Из наиболее известных можно назвать методику структурного анализа SADT и основанную на нем IDEF0, диаграммы потоков данных Гейна-Сарсона, методику объектно-ориентированного анализа UML, и др.

Другим важным моментом анализа предметной области является анализ процессов обработки данных. Для этого используют методику построения функциональной модели. Определим функциональную модель предметной области базы данных как совокупность некоторых моделей, предназначенных для описания процессов обработки информации.

1.4.1. Модель процесса

Модель процессов предназначена для описания процессов и функций системы в предметной области БД. Она чаще всего разрабатывается в соответствии с методологией IDEF0. В рамках этой методологии модель процесса представляется в виде совокупности иерархически упорядоченных и взаимосвязанных диаграмм (IDEF0-диаграмм). IDEF0-диаграммы включают в себя следующие диаграммы:

- контекстная диаграмма;
- диаграмма декомпозиции;
- диаграмма дерева узлов;
- диаграмма "только для экспозиции".

Контекстная диаграмма является началом иерархической структуры диаграмм и представляет собой самое общее описание системы. Она определяет взаимодействие анализируемой предметной области с внешней средой. Дальнейшее описание системы строится на основе последовательного разбиения функциональности системы на более детальные фрагменты. Диаграммы, которые описывают каждый из функциональных фрагментов системы, называются диаграммами декомпозиции.

Диаграмма дерева узлов показывает иерархическую структуру функций, не отображая взаимосвязи между ними.

Диаграммы "только для экспозиции" представляют, по сути, копии стандартных диаграмм, но не включаются в анализ синтаксиса модели. Они предназначены для демонстрации иных точек зрения на работы, для отображения деталей, которые не поддерживаются явно синтаксисом модели.

Основными элементами IDEF0-диаграмм являются процессы, входные и выходные параметры, управление, механизмы и вызов.

Процессы (Activity) обозначают процессы, функции или задачи, которые реализуются в течение определенного времени и имеют значимые предметной области результаты. Процессы изображаются в виде прямоугольников и именуются глаголом или отглагольными существительными. Взаимодействие процессов между собой и с внешними миром отражается стрелками. Стрелки (Arrow) именуются существительными и могут обозначать на диаграмме вход, выход, управление, механизм и вызов.

Вход (Input) – это материалы или информация, которые используются или преобразуются процессом для получения результата (выхода). Стрелка входа рисуется как стрелка, входящая в левую грань работы.

Управление (Control) – это правила, стратегии, процедуры или стандарты, которые ограничивают выполнение процесса. Она рисуется как стрелка, входящая в верхнюю грань процесса. Управление влияет на процесс, но не преобразуется им.

Выход (Output) – это материалы или информация, которые являются результатом процесса. Каждый процесс должен иметь хотя бы одну стрелку выхода, которая рисуется как стрелка, выходящая из правой грани процесса.

Механизм – это ресурсы, которые требуются для выполнения процесса (персонал, станки, устройства). Стрелка механизма рисуется как стрелка, входящая в нижнюю грань работы. Механизмы могут не изображаться.

Вызов (Call) – это специальная стрелка, указывающая на другую модель работы. Стрелка вызова рисуется как стрелка, исходящая из нижней грани работы. Стрелка вызова указывает, что некоторая работа выполняется за пределами моделируемой системы.

На рис. 1.1 приведен пример контекстной диаграммы процесса «Работа деканата».

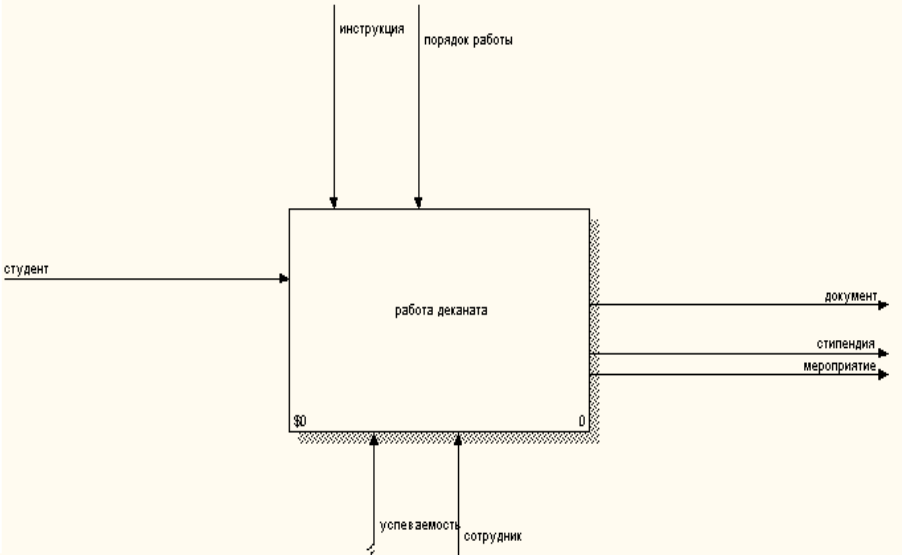

Рис. 1.1. Контекстная диаграмма процесса «Работа деканата»

1.4.2. Модель потока данных

Модель потока данных предназначена для описания процессов перемещения данных в предметной области БД. Модель потока данных представляется в виде диаграммы потока данных (Data Flow Diagram). Основными элементами диаграммы являются:

- источники данных (Data Source);
- процессы обработки данных (Data Process);
- хранилища данных (Data store);
- потоки данных (Data Flow).

Источники данных показывают, кто использует данные или работает с ними. Процессы обработки данных показывают операции, производимые над данными. Хранилища данных отражают места хранения данных. Потоки данных показывают способ передачи данных между источниками и хранилищами данных. Пример модели потока данных представлен на рис. 1.2.

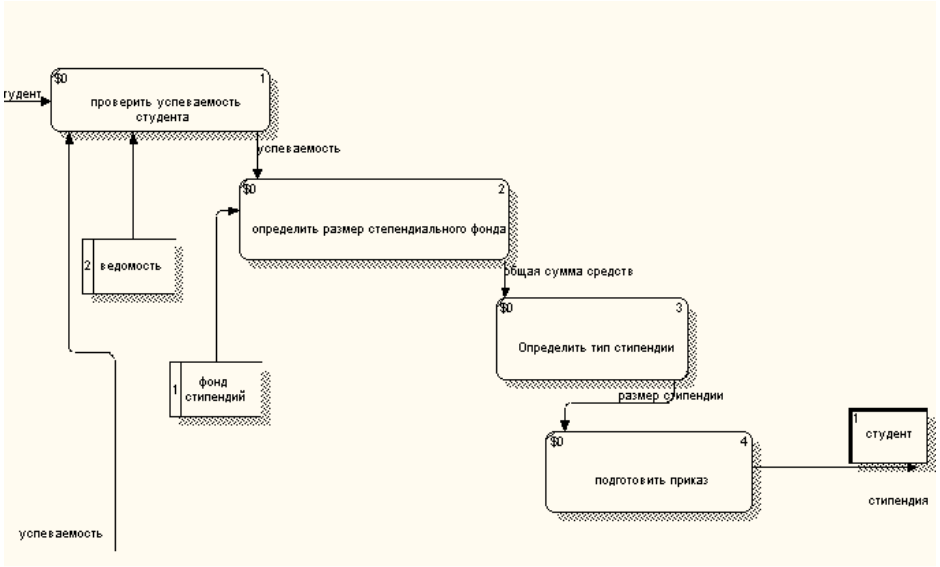

Рис. 1.2. Диаграмма потоков данных

Диаграмма потока данных позволяет:

- представить систему с точки зрения источников и потребителей данных;
- показать перемещение данных в процессе их обработки;
- показать внешние механизмы подачи данных;
- показать метод сбора данных.

Она предоставляет проектировщику баз данных информацию о хранилищах данных, что позволит на последующих стадиях проектирования обоснованно определить число баз данных для информационной системы; принятых схемах преобразования информации в процессе ее обработки, что позволит в ходе проектирования приложений составить спецификацию приложений.

1.5. Функции, структура и архитектура СУБД

Программы работы с данными:

- СУБД: ядро (создание БД, организация ввода-вывода, эффективная обработка и хранение данных),
- Приложения для решения конкретных задач.

Языковые средства: языки описания и обработки данных, языки программирования, языки для пользователей различных категорий (генераторы отчетов, инструментальные средства поддержки решений, генераторы приложений, языки высокого уровня).

Функции СУБД:

- Определение общей структуры БД и представления конкретных пользователей;
- Задание смысловых правил, контролирующих целостность данных;
- Определении правил безопасности данных;
- Поддержка операций работы с данными; добавление, изменение, удаление, поиск;
- Поддержка различных компьютерных платформ;
- Наличие средств администрирования БД;

- Обеспечение контролируемого доступа к данным за счет системы безопасности, предотвращающей несанкционированный доступ к данным;

- поддержка целостности непротиворечивости данных;

- управление параллельной работой приложений с помощью системы, контролирующей процессы совместного доступа к данным. Этот процесс несложно организовать, если разрешить только параллельное чтение данных, и становится достаточно сложным, если параллельно реализуются запросы на чтение и обновление или удаление одних и тех же данных;

- управление буферами оперативной памяти – для ускорения процессов работы с БД СУБД поддерживает собственный набор буферов в оперативной памяти;

- поддержка обмена данными при работе в сети;

- поддержка независимости данных;

- управление транзакциями - транзакции могут содержать сразу несколько операций с данными и быть достаточно сложными)

- наличие системы восстановления, позволяющей восстановить БД до предыдущего непротиворечивого состояния (точки отката);

- наличие каталога хранимой в БД информации – с помощью интегрированного системного каталога – словаря БД. В словаре содержится информация об именах, типах и размерах хранимых данных, имена связей, накладываемые ограничения поддержки целостности, имена пользователей и их права доступа к данным, отображения между различными уровнями представления данных (внешняя, концептуальная и внутренняя схемы и отображения между ними), статистически данные (частота транзакций, счетчики обращений к объектам БД и др.);

- вспомогательные функции – обычно дополнительные функции для администрирования, которые реализуются в виде утилит.

1.5.1. Архитектура системы БД

В современных системах БД выделяют 3 уровня представления данных: внешний, концептуальный и внутренний (рис. 1.3). *Внешний уровень* отражает представление пользователей на БД. *Внутренний уровень* отражает представление, на котором СУБД и ОС воспринимают данные. *Концептуальный (логический) уровень* – связан с обобщенным представлением всех пользователей БД (это взгляд администратора БД).

Рис. 1.3. Уровни представления данных в БД

Этот подход является общепризнанным и существует уже более 30 лет. Первые предложения по многоуровневой архитектуре были выдвинуты рабочей группой CODACYL в 1971 г. В 1975 г. Комитет планирования стандартов и норм SPARC (Standarts Planning and Requirements Committee) Американского национального института стандартов FNSI (American National Standrt Institute) предложил обобщенную 3-х уровневую архитектуру БД, которая была официально признана в 1978 г и получила название ANSI SPARC.

Первая попытка создания стандартной терминологии и общей архитектуры СУБД была предпринята в 1971 году группой DBTG, признавшей необходимость использования двухуровневого подхода, построенного на основе использования системного представления, т.е. схемы, и пользовательских представлений, т.е. подсхем. Сходные терминология и архитектура были предложены в 1975 году Комитетом планирования стандартов и норм SPARC. Комитет ANSI/SPARC признал необходимость использования трехуровневого подхода при определении архитектуры СУБД. Хотя модель ANSI/SPARC не стала стандартом, тем не менее, она все еще представляет собой основу для понимания некоторых функциональных особенностей СУБД. Цель трехуровневой архитектуры заключается в отделении пользовательского представления базы данных от ее физического представления. Ниже перечислено несколько причин, по которым желательно выполнять такое разделение.

- Каждый пользователь должен иметь возможность обращаться к одним и тем же данным, используя свое собственное представление о них. Каждый пользователь должен иметь возможность изменять свое представление о данных, причем это изменение не должно оказывать влияния на других пользователей.
- Пользователи не должны непосредственно иметь дело с подробностями физического хранения данных в базе
- Администратор базы данных должен иметь возможность изменять структуру хранения данных в базе, не оказывая влияния на пользовательские представления.
- Внутренняя структура базы данных не должна зависеть от таких изменений физических аспектов хранения информации, как переход на новое устройство хранения.
- АБД должен иметь возможность изменять концептуальную или глобальную структуру базы данных без какого-либо влияния на всех пользователей.

Уровень, на котором воспринимают данные пользователи, называется внешним уровнем (external level), тогда как СУБД и операционная система воспринимают данные на внутреннем уровне (internal level). Именно на внутреннем уровне данные реально сохраняются с использованием всех тех структур и файловой организации. Концептуальный уровень (conceptual level) представления данных предназначен для отображения внешнего уровня на внутренний и обеспечения необходимой независимости уровней друг от друга.

Основным назначением такой архитектуры является обеспечение независимости данных, суть которого заключается в том, что изменения на нижних уровнях архитектуры не влияют на верхние уровни.

Внешний уровень – это пользовательский уровень. Пользователем может быть программист, конечный пользователь или администратор БД. Представление с точки зрения пользователя называется внешним представлением. Оно отражает представление пользователя о данных в БД в удобной для него форме (например, для бухгалтерии - это данные о студентах, которым начисляется стипендия, при этом не важно, какие у них оценки). Для деканата данные о тех же студентах будут

содержать сведения об их оценках в сессию и т.п. Эти частичные представления пользователей о БД называют подсхемой БД. При этом каждый пользователь может использовать свои языковые и программные средства для работы с БД.

Концептуальный уровень является промежуточным уровнем и обеспечивает представление всей БД в абстрактной форме. Описание БД на этом уровне называют концептуальной схемой, которая является результатом концептуального проектирования. Концептуальная схема отражает интересы всех пользователей и представляет собой единое логическое описание всех элементов данных и отношений между ними, образуя логическую структуру всей БД.

Концептуальная схема должна содержать:

- объекты и атрибуты предметной области;
- связи между объектами;
- ограничения, накладываемые на данные;
- семантическую информацию о данных;
- обеспечение безопасности данных;
- поддержку целостности данных.

Концептуальный уровень поддерживает каждое внешнее представление, в том смысле, что любые доступные пользователю данные должны содержаться (или могут быть вычислены) на этом уровне. Однако на этом уровне не содержаться сведения о методах хранения данных.

Внутренний уровень характеризует внутреннее представление. Оно не связано с физическим уровнем, т.к. физический уровень хранения информации обладает существенной индивидуальностью для каждой системы. На внутреннем уровне все эти особенности не учитываются, а вся область хранения представляется как бесконечное линейное адресное пространство. На нижнем уровне находится внутренняя схема, которая является полным описанием внутренней модели данных. Для каждой БД она только одна. Внутренняя схема описывает физическую реализацию и предназначена для достижения оптимальной производительности и обеспечения экономного использования дискового пространства. На внутреннем уровне хранится следующая информация:

- распределение дискового пространства для хранения данных и индексов;
- описание подробностей сохранения записей (типы, размеры элементов данных и др.);
- сведение о размещении записей;

- сведения о сжатии записей и выбранных методах шифрования.

СУБД отвечает за установление соответствия между всеми тремя уровнями и поддержку их непротиворечивости.

1.5.2. Структура современной СУБД

СУБД является сложным программным продуктом и должна обеспечивать выполнение всех определенных ранее функций. Следует отметить, что некоторые функции могут выполняться ОС. Поэтому при проектировании СУБД следует учитывать особенности интерфейса СУБД с ОС.

В структуре СУБД выделяют ядро СУБД (рис. 1.4), которое включает транслятор с языка БД (ЯОД И ЯМД), подсистему поддержки времени выполнения (управление данными во внешней памяти, управление собственными буферами оперативной памяти, управление журнализацией), подсистему управления транзакциями. Кроме того в состав СУБД входит набор утилит, который существенно зависит от выбранной СУБД. В некоторых СУБД эти части явно не выделены, но логически присутствуют всегда.

Ядро СУБД отвечает за управление данными во внешней памяти, управление буферами ОП, управление транзакциями и ведение журналов. Каждая функция реализуется в виде соответствующего менеджера (менеджер транзакций, менеджер буферов и т.д.). Эти функции взаимосвязаны и должны работать под управлением четко проверенных протоколов. Ядро СУБД обладает собственным интерфейсом, недоступным пользователю напрямую и является резидентной частью СУБД. При использовании архитектуры «клиент-сервер» оно является основной составляющей серверной части. Поэтому программа, обеспечивающая выполнение функций СУБД называется машина БД.

В структуре программного комплекса СУБД можно выделить:

Процессор запросов – преобразует запросы в последовательность низкоуровневых инструкций для контроллера БД.

Транслятор с ЯОД – преобразует его команды в набор таблиц, содержащих метаданные. Эта информация хранится в системном каталоге, а управляющая информация – в заголовке файлов.

Транслятор с ЯМД – преобразует внедренные в прикладные программы операторы в вызовы стандартных функций базового языка. Взаимодействует с процессором запросов.

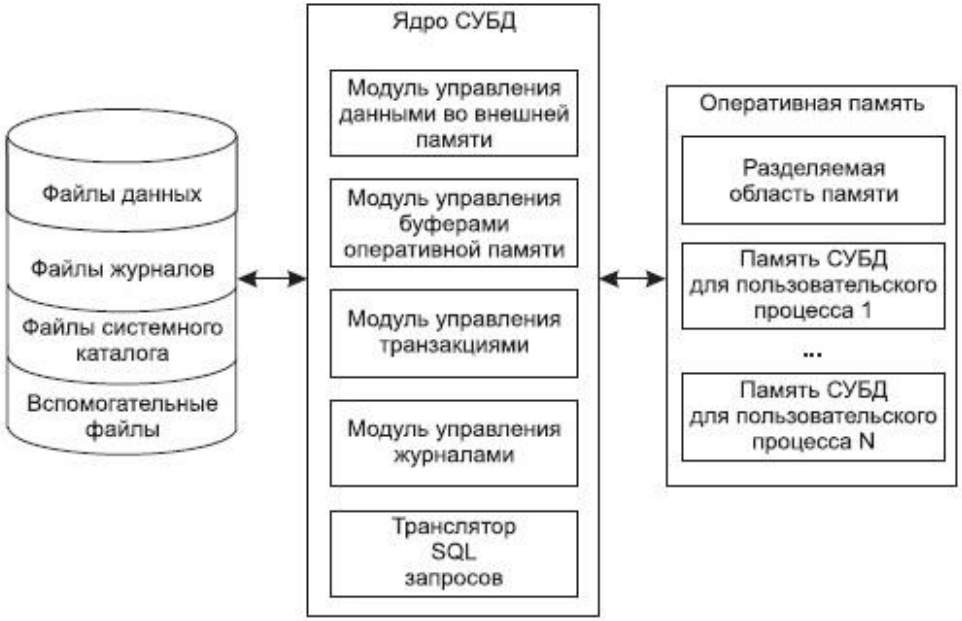

Рис. 1.4. Структура СУБД

Контроллер словаря – обеспечивает доступ к системному каталогу и работу с ним.

Контроллер файлов – манипулирует фалами с данными и отвечает за распределение дискового пространства. Он не управляет физическим вводом/выводом, а передает запросы соответствующим методам доступа ОС.

Контроллер БД – взаимодействует с запущенными пользователями программами и запросами. Он принимает запросы и проверяет внешние и концептуальные схемы для определения тех концептуальных записей, которые необходимы для выполнения запроса к БД. Затем контроллер БД вызывает контроллер файлов для выполнения запроса. В состав контроллера БД входят следующие программные компоненты:

- контроль прав доступа – проверяет наличие у данного пользователя полномочий на выполнения затребованной операции;
- процессор команд – выполняет запрос;

- средства контроля целостности – осуществляют проверку ограничений поддержки целостности при выполнении операций изменения данных;
- оптимизатор запросов – определяет оптимальную стратегию выполнения запроса;
- контроллер транзакций – осуществляет обработку операций, поступающих в процессе транзакции;
- планировщик – отвечает за бесконфликтное выполнение параллельных операций с БД и управляет относительным порядком выполнения операций, определенным в разных транзакциях;
- контроллер восстановления – отвечает за восстановление БД при сбоях до непротиворечивого состояния;
- контроллер буфера – отвечает за перенос данных между оперативной памятью и жестким диском.

1.5.3. Языки СУБД

Многие СУБД поддерживают возможность внедрения собственных операторов в языки высокого уровня (COBOL, Fortran, Pascal, Adam, C). Но в СУБД поддерживается 2 специализированных языка для разработки приложений с БД – ЯОД (DDL – Data Definition Language) и ЯМД (DMD – Data Manipulation Language). *ЯОД* – описательный язык для определения логической схемы БД. Он состоит из набора операторов, задающих определение схемы БД. Результатом компиляции таких описаний (концептуальной и внешних схем) является набор таблиц, хранимый в специальном системном каталоге БД. (Он хранит метаданные). ЯОД не используется для работы с данными. Для этого используется *ЯМД*, который содержит набор операторов, выполняющих обработку данных: поиск, добавление, изменение и удаление. Поддержка ЯМД – одна из основных функций СУБД. Различают 2 подхода к реализации ЯМД: процедурный и декларативный.

При использовании *процедурного ЯМД* пользователь определяет последовательность действий, которые необходимо выполнить, чтобы получить требуемый результат. Этот поход аналогичен тому, как реализованы процедурные языки программирования (Pascal, C и др.). ЯМД в этом случае предоставляет пользователю набор операторов над данными. К этому типу относятся языки, основанные на реляционной алгебре.

Декларативные ЯМД позволяют определить весь требования к результирующим данным с помощью одного оператора. В этом случае нет необходимости знать детали внутренней реализации структур данных и особенности алгоритмов, используемых для их извлечения. К этому типу относятся языки, основанные на реляционном исчислении SQL и QBE. *SQL* (Structured Query Language) основан на реляционном исчислении и поддерживается всеми реляционными СУБД. Для него определен международный стандарт. *QBE* (Query By Example) – простой язык с графическим интерфейсом, который позволяет формулировать запрос непрофессиональным пользователям (например, в СУБД Access).

Современные СУБД содержат также в своем составе набор инструментальных средств, облегчающих разработку приложений с БД – различного рода генераторы:

- экранных форм – для создания шаблонов ввода данных;
- отчетов;
- графического представления данных в виде диаграмм;
- приложений для создания программ обработки данных.

1.6. Модели данных

Моделирование является одним из основных способов описания БД. В основе любой СУБД, как правило, лежит единый подход к логическому представлению данный, определяемый её логической моделью. Различают следующие виды логических моделей БД:

- Иерархическая
- Сетевая
- Реляционная
- Постреляционная
- Многомерная
- Объектно-ориентированная и др.

1.6.1. Иерархическая модель данных

Иерархическая модель относится к графовым (рис. 1.5) моделям и представляет собой совокупность элементов, связанных между собой по определенным правилам. Графическим способом представления иерархической структуры является дерево.

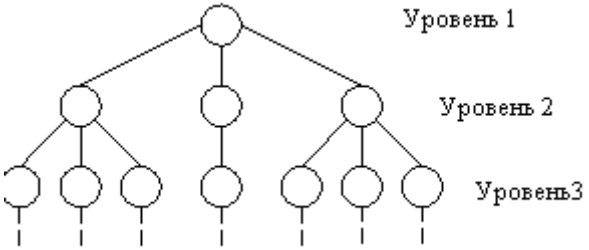

Рис. 1.5. Графическое представление древовидной структуры данных

Дерево представляет собой иерархию элементов, называемых узлами. Элемент представляет собой совокупность атрибутов, описывающих объекты предметной области. В иерархической модели выделяют корневой узел (корень дерева), который находится на самом верхнем уровне и не имеет узлов, стоящих выше него. У одного дерева может быть только один корень. Остальные узлы, называемые порожденными, связаны между собой по правилу: каждый узел имеет только один исходный, находящийся на более высоком уровне, и любое число подчиненных узлов на следующем уровне. БД в этом случае представляется как совокупность деревьев (лес).

Примером простого иерархического представления могут служить (рис. 1.6) объекты предметной области: административная структура высшего учебного заведения (университет, институты в составе университета, студенческая группа).

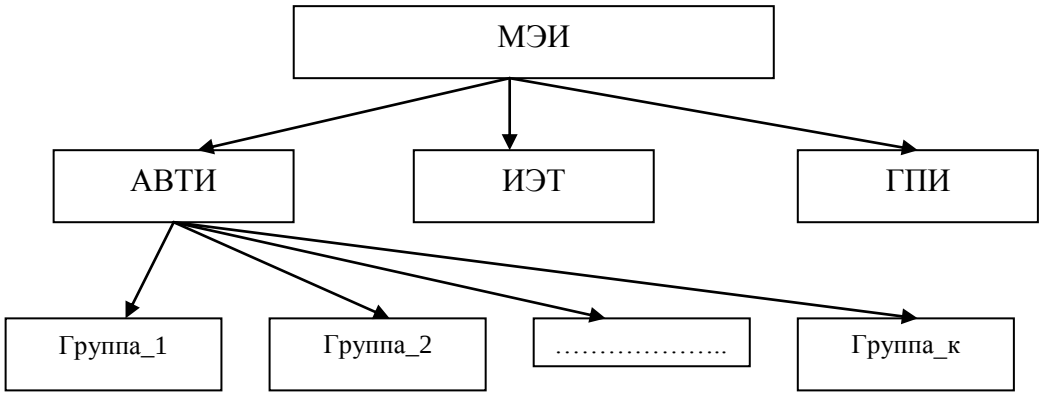

Рис. 1.6. Пример иерархической структуры

К *достоинствам* иерархической модели данных относятся эффективное использование памяти ЭВМ и неплохие показатели времени выполнения операций поиска данных.

Недостатком иерархической модели является ее неэффективность при обработке информации с достаточно сложными логическими связям, сложность изменения логической структуры данных, проблемы, связанные с модификацией данных.

На иерархической модели данных были основаны первые СУБД, такие как IMS , PC Focus , Ока, ИНЭС и МИРИС. В настоящее время эта модель данных практически не используется при построении универсальных СУБД, но применяется при создании специализированных СУБД.

1.6.2. Сетевая модель данных

В отличие от иерархической модели сетевые структуры данных (рис. 1.7, 1.8) поддерживают множественные связи элементов. Каждый элемент в сетевой структуре может быть связан с любым другим элементом.

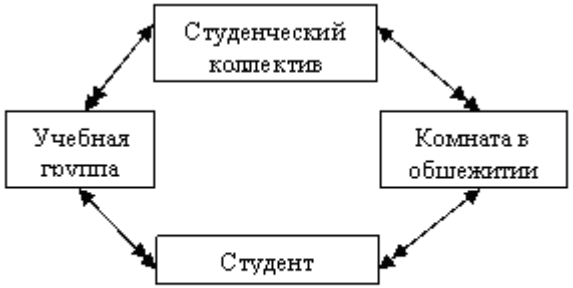

Рис. 1.7. Сетевая модель

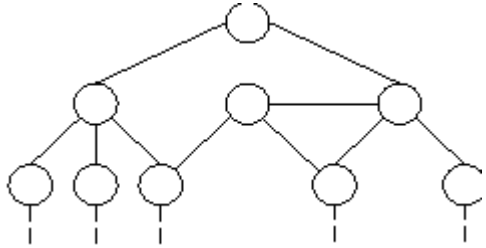

Рис. 1.8. Сетевая структура данных

Достоинством сетевой модели данных является возможность эффективной реализации с точки зрения использования внешней памяти и времени выполнения поисковых запросов.

Недостатком сетевой модели данных являются высокая сложность её реализации, схема БД является жесткой, её изменении приводит к серьёзным перестройкам БД, построенной на ее основе.

Наиболее известными сетевыми СУБД являются CODASYL, СЕДАН, db _ VistaIII, СЕТОР и КОМПАС.

В настоящее время подавляющее большинство СУБД поддерживают реляционную модель.

Контрольные вопросы

1. Назовите классы пользователей БД.
2. Перечислите свойства БД.
3. Определите понятие логической и физической независимости данных.
4. Для чего используется модель IDEF0?
5. Дайте характеристику основных элементов IDEF0-диаграмм.
6. Для чего используется DFD-модель?
7. Перечислите функции СУБД.
8. Дайте характеристику уровней архитектуры системы БД.
9. Назовите компоненты структуры СУБД.
10. Определите особенности иерархической модели БД.
11. Перечислите виды логических моделей БД.

ЛЕКЦИЯ 2. ПРОБЛЕМЫ ПРОЕКТИРОВАНИЯ БД

Жизненный цикл БД. Проблемы проектирования. Этапы проектирования БД. Концептуальная (инфологическая) модель. ER-модель.

2.1 Понятие жизненного цикла БД

Жизненный цикл БД определяется как совокупность этапов, начиная с момента возникновения идеи о создании БД и заканчивая выводом её из эксплуатации. Этапы жизненного цикла БД:

- возникновение и исследование идеи;
- анализ требований и проектирование;
- реализация; ввод в эксплуатацию;
- эксплуатация и сопровождение;
- завершение эксплуатации.

В зависимости от последовательности выполнения этапов различают следующие модели жизненного цикла:

- каскадная,
- итерационная
- спиральная.

Каскадная модель жизненного цикла (рис. 2.1.) характеризуется строго последовательным выполнением этапов жизненного цикла. Его можно рекомендовать только для сравнительно небольших БД, для которых точно определены все требования. Она удобна тем, что можно заранее определить сроки и стоимость реализации БД.

Рис. 2.1. Каскадная модель жизненного цикла

В итерационной модели (рис. 2.2) допускается возврат к предыдущим этапам жизненного цикла, что делает эту модель сложной для управления.

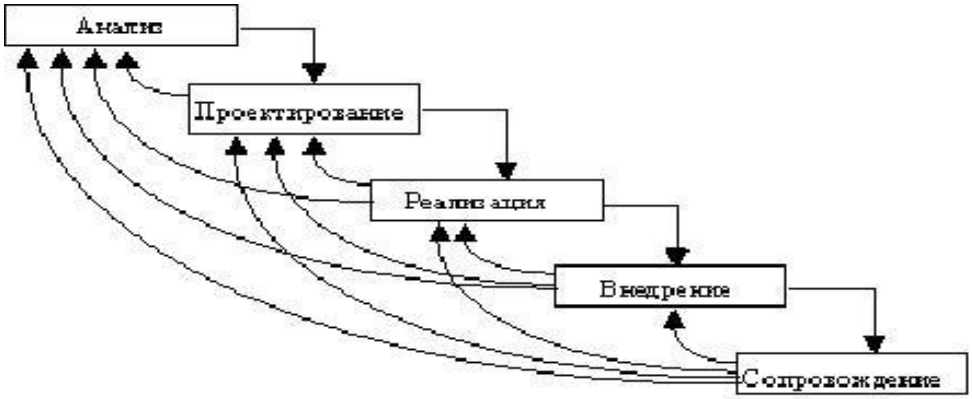

Рис. 2.2. Итерационная модель жизненного цикла

В большинстве случаев жизненный цикл БД реализуется на основе спиральной модели (рис. 2.3), которая обеспечивает возможность поддержки различных версий сложного программного продукта, к которым относятся БД и СУБД. Эта модель жизненного цикла используется при разработке практически любого сложного программного продукта.

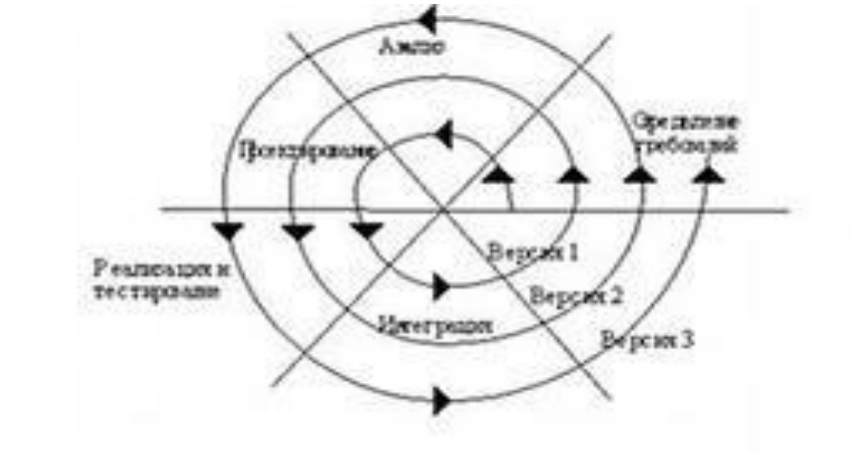

Рис. 2.3. Спиральная модель жизненного цикла

2.2 Проблемы проектирования РБД

Проектирование БД – одна из наиболее сложных и важных задач, в результате решения которой должны быть определены содержание БД, эффективный для всех её будущих пользователей способ организации данных и инструментальные средства управления данными.

При разработке больших систем проектирование БД требует особой тщательности, т.к. цена допущенных на этой стадии просчётов и ошибок особенно велика. Основной целью процесса проектирования БД является разработка такого проекта, который удовлетворяет следующим основным требованиям:

- Корректность схемы БД, т.е. база должна адекватно отображать предметную область. При этом каждому объекту предметной области должен соответствовать набор данных в БД, а каждому процессу его обработки – адекватные процедуры работы с ними.
- Обеспечение ограничений на объёмы внешней и оперативной памяти и другие ресурсы вычислительной системы.
- Эффективность функционирования, которое определяет необходимость выполнения временных ограничений на запрос к данным.
- Простота и удобство эксплуатации.
- Возможность развития и адаптации БД к изменениям в предметной области и/или требований пользователей.

В процессе проектировании базы данных решаются две основные проблемы.

1. Каким образом отобразить объекты предметной области в абстрактные объекты модели данных таким образом, чтобы это отображение отражало семантику предметной области и было, по возможности, лучшим в каком-то смысле (эффективным, удобным и т. д.)? Для успешного решения этой проблемы необходимо выполнить этапы инфологического и логического проектирования БД.
2. Как обеспечить эффективность выполнения запросов к БД? При этом необходимо обеспечить эффективное хранение данных во внешней памяти и оптимальное (сточки зрения времени выполнения) выполнение запросов к БД. Для этого может потребоваться создание дополнительных структур данных (например, индексов). Решение этой проблемы связывают с физическим проектированием БД.

Таким образом, в процессе проектирования БД можно выделить следующие этапы:

1. *Инфологическое проектирование*. На этом этапе изучается предметная область, в которой будет использоваться БД, и информационные потребности будущих пользователей. Другая задача этого этапа – анализ предметной области, который позволяет сформировать взгляд на неё с позиций сообщества будущих пользователей БД. Для этого на этапе инфологического проектирования решаются следующие задачи:

- обследование предметной области, изучение ее информационной структуры;
- выявление всех фрагментов, каждый из которых характеризуется пользовательским представлением, информационными объектами и связями между ними, процессами с информационными объектами;
- моделирование и интеграция представлений всех пользователей БД.

Результатом данного этапа является *инфологическая модель* предметной области, которая реализуется с помощью формализованных методов: семантических сетей, ER-моделей и др.

2. *Логическое проектирование* связано с построением концептуальной модели БД на основе выбранной логической модели данных. Оно выполняется с помощью преобразования инфологической модели в структуры данных. В результате формируется общая структура БД и спецификации прикладных программ. На этом этапе часто моделируют базы данных применительно к различным СУБД и проводят сравнительный анализ моделей.
3. Физическое проектирование связано с определением особенностей хранения данных, методов доступа и т.д. Решение этой задачи в настоящее время может быть сведено к выбору СУБД.

2.3 Инфологическое моделирование БД

Инфологическая модель - формализованное описание предметной области. Она представляет собой описание предметной области, которое основано на анализе семантики объектов и явлений предметной области, и выполнено без ориентации на использование в дальнейшем программных или технических компьютерных средств. Она реализует такое формализованное описание предметной области, которое легко будет «читаться» не только специалистами по БД, но и специалистами, которые будут использовать БД. Это описание должно быть настолько емким, чтобы можно было оценить глубину и корректность проработки проекта БД не должно быть привязано к конкретной СУБД. Инфологическое проектирование, прежде всего, связано с необходимостью представления семантики предметной области в модели БД, Поэтому для ее описания используются семантические (смысловые) модели. Для построения инфологической модели необходимо провести обследование предметной области, которое заключается в выделении информационных объектов, данные о которых будут храниться в БД, изучении документов, циркулирующих в системе на основе интервьюирования работников и изучения различной документации и многое другое. Построение инфологической модели может осуществляться вручную или с использованием специальных CASE-средств.

Основные компоненты инфологической модели:

- описание объектов предметной области и связей между ними;
- описание информационных потребностей пользователей;
- описание существующей информационной системы (документы, документооборот, существующей АИС);
- описание алгоритмических зависимостей показателей;
- описание ограничений целостности;
- описание функциональной структуры системы, для которой создается БД;
- требования к ИС и существующие ограничения;
- требования к интерфейсу.

Требования к инфологической модели:

- адекватное отображение предметной области – отражение всей необходимой информации и связей между ее элементами;
- непротиворечивость - учет информационных потребностей многих пользователей и взаимосвязей между данными;

- однозначность трактовки модели всеми её пользователями – обеспечивается использованием формализованного языка;
- легкость восприятия всеми категориями пользователей;
- конечность модели – описание только выделенной предметной области;
- простота модификации – простота внесения изменений в модель на уровне данных (добавление и удаление её компонентов);
- возможность декомпозиции и композиции модели.

Проблема представления семантики давно интересовала разработчиков, и в 70-х годах прошлого века было предложено несколько таких моделей:

- семантические сети;
- семантическая модель данных, предложенная Хаммером (Hammer) и Мак-Леоном (McLean) в 1981 году;
- функциональная модель данных Шипмана (Shipman), созданная в 1981 году;
- модель «сущность-связь» (ER-модель), разработанная Ченом (Chen) в 1976 году.

В практике современной разработки БД именно ER-модель стала негласным стандартом при инфологическом моделировании БД. В сочетании с соответствующими CASE-средствами этот вид инфологической модели обладает следующими преимуществами:

- делает анализ предметной области более конкретным и целенаправленным за счет использования методологии проектировании;
- является удобным средством документирования модели; повышение качества документирования проекта при использовании CASE-средств;
- позволяет вести проектирования АИС без привязки к конкретной СУБД;
- снижают требования к знанию языков конкретных СУБД;
- позволяют осуществлять простой переход при смене СУБД;
- осуществляют прямое и обратное проектирование;
- сокращают время проектирования системы;

- автоматизированное тестирование проекта на всех этапах проектирования;
- вести коллективную разработку проекта.

2.3.1. ER-модели

Основными базовыми понятиями ER-модели являются сущность, связь, атрибут. ***Сущность*** (Entity) — множество экземпляров реальных или абстрактных объектов предметной области (людей, событий, состояний, идей, предметов и др.), которые обладают общими свойствами (*атрибутами)*. Различают *сущности-понятия* и *сущности-экземпляры*. Сущность-понятие (в дальнейшем просто сущность) определяет некоторое множество объектов предметной области с одинаковыми свойствами (например, сущность Студент, Дисциплина и др.). В этом случае можно сказать, что сущность определяет некоторый тип сущности. Любой объект предметной области может быть представлен только одной сущностью, которая должна быть уникально идентифицирована. При этом имя сущности должно отражать тип или класс объекта, а не его конкретный экземпляр (например, Студент, а не Иванов).

Каждая сущность-экземпляр должна обладать уникальным *идентификатором – ключом***,** который позволяет однозначно определить его и отличать от всех других экземпляров данного типа сущности. Каждая сущность-понятие должна обладать следующими свойствами:

- иметь уникальное имя;
- к одному и тому же имени должна всегда применяться одна и та же интерпретация;
- одна и та же интерпретация не может применяться к различным именам, если только они не являются псевдонимами;
- иметь один или несколько атрибутов, которые либо принадлежат сущности, либо наследуются через связь;
- иметь один или несколько атрибутов, которые однозначно идентифицируют каждый экземпляр сущности.

При разработке инфологической модели моделируются именно сущности-понятия.

Примеры сущностей: студент, учебная дисциплина, преподаватель. Экземплярами сущностей являются конкретный студент, конкретная учебная дисциплина, конкретный преподаватель.

Каждая сущность может обладать любым количеством связей с другими сущностями модели.

Связь (Relationship) – поименованная ассоциация между двумя *сущностями*, значимая для рассматриваемой предметной области. *Связь* – это ассоциация между сущностями, при которой каждый экземпляр одной сущности ассоциирован с

произвольным (в том числе нулевым) количеством экземпляров другой сущности. Она показывает, как в предметной области информационные объекты взаимодействуют друг с другом. Например, преподаватель ведет занятия по учебной дисциплине; студент изучает дисциплину. Между двумя сущностями может быть определено несколько связей, каждая из которых обладает своей семантикой. Наличие множества связей и определяет сложность инфологических моделей. При построении ER-модели используются бинарные связи, т.е. связи между двумя сущностями. В этом случае можно выделить родительскую сущность (откуда выходит связь) и дочернюю сущность. Важной характеристикой является мощность связи (кардинальность или степень связи), которая показывает, сколько экземпляров дочерней сущности связано с одним экземпляром родительской сущности. Различают следующие типы связей:

- связь один-к-одному,
- связь один-ко-многим,
- связь многие-ко-многим.

Связь ОДИН-К-ОДНОМУ (1:1) определяет то, что в каждый момент времени каждому экземпляру сущности *A* соответствует 1 или 0 экземпляров сущности *B* (рис.2.1). Например, такая связь отражает тот факт, что студент в конкретное время занимается только в одной аудитории.

Рис. 2.1.Связь 1:1

Связь ОДИН-КО-МНОГИМ (1:M) означает, что одному экземпляру сущности *A* соответствуют несколько экземпляров сущности *B* (рис. 2.2.). Например, такая связь отражает тот факт. Что преподаватель преподает дисциплину.

Рис. 2.1. Связь 1:M

Связь МНОГИЕ-КО-МНОГИМ (M:N) означает, что многим экземплярам сущности *A* соответствует много экземпляров сущности *B* (рис. 2.3). Например, многие студенты обучаются у одних и тех же преподавателей.

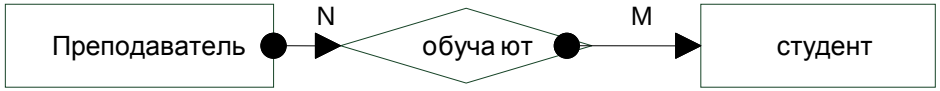

Рис. 2.2. Связь N:M

Все связи требуют описания, которое включает:

- идентификатор связи;
- формулировку имен связи с точки зрения связываемых сущностей;
- тип связи.

Связь может быть обязательной (если в данной связи должен участвовать каждый экземпляр сущности), и необязательной (если не каждый экземпляр сущности должен участвовать в данной связи). В ER-модели допускаются рекурсивные связи.

Введенное понятие связи между сущностями позволяет более детально рассмотреть, какие виды сущностей можно выделить в предметной области. Различают следующие понятия:

Сильная сущность (стержневая) – сущность, независящая от других сущностей предметной области. Например, сущности клиент, товар.

Слабая сущность – сущность, которая не может существовать без связи с другой сущностью. Например, сущность заказ не может существовать без сущности клиент.

Иногда выделенная сущность несет в себе отношение включения или часть-целое. При этом существует некоторый атрибут, значения которого определяет разбиение множества экземпляров сущности на непересекающиеся подмножества - категории сущности. Категории сущности называются подтипами и выделяются в подчиненную в рамках отношения сущность. Иногда из исходной сущности выделяются общие для полученных категорий атрибуты, и таким образом выделяется сущность, которая становится супертипом. За выделенной сущностью-супертипом обычно оставляют наименование исходной сущности, хотя ее семантический смысл меняется.

Супертип с порожденными им подтипами является примером, так называемой *составной сущности*. Составная сущность является логической конструкцией модели для представления набора сущностей и связей между ними как единого целого.

Атрибут (Attribute) – любая характеристика сущности, значимая для рассматриваемой предметной области и предназначенная для квалификации, идентификации, классификации, количественной характеристики или выражения состояния сущности. Атрибут представляет тип характеристик или свойств, ассоциированных с множеством реальных или абстрактных объектов (людей, мест, событий, состояний, идей, предметов и т.д.). *Экземпляр атрибута* – это определенная характеристика отдельного элемента множества. Экземпляр атрибута определяется типом характеристики и ее значением, называемым *значением атрибута*. На диаграмме «сущность-связь» атрибуты связаны с конкретными сущностями. Таким образом, экземпляр сущности должен обладать единственным определенным значением для связанного с ним атрибута.

Различают:

1. *Идентифицирующие и описательные атрибуты*. Идентифицирующие атрибуты имеют уникальное значение для сущностей данного типа и являются *потенциальными ключами.* Они позволяют однозначно распознавать экземпляры сущности. Из потенциальных ключей выбирается один первичный ключ (ПК). В качестве ПК обычно выбирается потенциальный ключ, по которому чаще происходит обращение к экземплярам записи. Кроме того, ПК должен включать в свой состав минимально необходимое для идентификации количество атрибутов. Остальные атрибуты называются описательными и заключают в себе интересующие свойства сущности.

2. *Составные и простые атрибуты*. Простой атрибут состоит из одного компонента, его значение неделимо. Составной атрибут является комбинацией нескольких компонентов, возможно, принадлежащих разным типам данных (например, ФИО или адрес). Решение о том, использовать составной атрибут или разбивать его на компоненты, зависит от характера его обработки и формата пользовательского представления этого атрибута.

3. *Однозначные и многозначные атрибуты* (могут иметь соответственно одно или много значений для каждого экземпляра сущности).

4. *Основные и производные атрибуты*. Значение основного атрибута не зависит от других атрибутов. Значение производного атрибута вычисляется на основе значений других атрибутов (например, возраст студента вычисляется на основе даты его рождения и текущей даты).

2.3.2. Этапы инфологического моделирования

Разработка инфологической модели БД проводится на основе данных, собранных в процессе анализа предметной области. Эти данные, как правило, представляют собой текстовое описание информационных потребностей группы пользователей,

которые будут использовать БД. Инфологическая модель используется на ранних стадиях разработки проекта. Она доступна для анализа программистам-разработчикам, которые будут разрабатывать отдельные приложения. Она имеет однозначную интерпретацию, в отличие от некоторых предложений естественного языка. Это и определило тот факт, что именно этот вид семантической модели является основным при проектировании БД.

Выделим основные шаги построения инфологической модели:

1. Выделить информационные объекты предметной области. На этом шаге определяются информационные объекты предметной области, существенные для поддержки решения задач пользователей. В качестве самостоятельной сущности следует отображать объекты, для которых:
    - фиксируются какие-либо атрибуты;
    - можно определить более чем в одну связи.
2. Определить ключевой атрибут для каждой сущности. В качестве ключевого атрибута следует определять тот, который обладает свойством уникальности значения для каждой отдельной сущности-экземпляра.
3. Определить связи между сущностями, которые задают смысловое взаимодействие сущностей.

Преимуществом ER-модели является то, что для нее существует алгоритм однозначного преобразования ее в реляционную модель данных. Это позволило в дальнейшем разработать большое количество инструментальных систем, которые поддерживающих процесс разработки БД. Одним из таких инструментальных средств является CASE-средства Erwin Data Modeler.

Контрольные вопросы

1. Что такое жизненный цикл БД?
2. Дайте характеристику основных моделей жизненного цикла БД
3. Назовите основные проблемы проектирования БД
4. Охарактеризуйте основные этапы проектирования БД
5. Для чего используется инфологическая модель БД?
6. Дайте характеристику ER-модели
7. Какие типы связей различают в ER-модели?
8. Что такое ключевой атрибут?
9. Какие виды сущностей различают в ER-модели?
10. Какие виды атрибутов связей различают в ER-модели?
11. Что определяет мощность связи?

ЛЕКЦИЯ 3. ТЕОРИЯ РЕЛЯЦИОННОЙ МОДЕЛИ БД

Теоретические основы реляционной модели данных (РМД). Основные элементы РМД: отношение, ключ, связь. Реляционная алгебра. Полная система операций реляционной алгебры. Языки манипулирования, основанные на реляционной алгебре и на исчислении отношений

Реляционная модель – это абстрактная теория данных, в основу которой полжены разделы математики: теория множеств и логика предикатов. Принципы реляционной модели были сформулированы в 1969 и 1970 годах Е.Ф. Коддом (E.F. Codd), который в то время работал в корпорации IBM. Он, будучи математиком по образованию, в 1968 году предложил применять для решения задач управления БД строгие и точные математические принципы. Свои идеи впервые подробно изложил в статье "A Relational Model of Data for Large Shared Data Banks", ставшей классической. С этого момента начались интенсивные работы по развитию теории реляционной модели данных (РМД), которые оказали заметное влияние на другие области информационных технологий, такие как искусственный интеллект, обработка естественных языков и проектирование аппаратных средств.

3.1. Основные понятия

Фундаментальным понятием реляционной модели данных является понятие **отношения**. Кодд доказал, что любое представление данных может быть сведено к совокупности двумерных таблиц особого вида – *отношений* (relation).

Формально, отношение определяется как подмножество декартового произведения образующих его множеств (*доменов*):

$R \subseteq D_1$ x D_2 x … x D_n, где D_i – i-ый домен отношения.

Таким образом, отношение представляет собой множество *кортежей* (n-ок) вида $(a_1, a_2, \ldots, a_n)$, где каждый $a_i \in D_i$. Дадим еще несколько определений.

Определение 1. Атрибут отношения есть пара вида <Имя_атрибута : Имя_домена>.

Имена атрибутов должны быть уникальны в пределах отношения.

Определение 2. Отношение R, определенное на множестве доменов D_1, D_2 …D_n (не обязательно различных), содержит две части: заголовок и тело.

Заголовок отношения (или *схема отношения*) содержит фиксированное количество атрибутов отношения:

$$(< A_1 : D_1 >, < A_2 : D_2 >, \ldots, < A_n : D_n >)$$

Тело отношения содержит множество кортежей отношения. Каждый *кортеж отношения* представляет собой множество троек вида <Имя_атрибута : Тип_атрибута:Значение_атрибута>:

(<A_1:T_1:Val_1>, A_2:T_2:Val_2>,…,<A_n:T_n:Val_n>)

Отношение обычно записывается в виде:

$$R(< A_1 : D_1 >,< A_2 : D_2 >,\ldots,< A_n : D_n >),$$

или короче

$$R(A_1, A_2, \ldots, A_n).$$

Полем называется значение атрибута в кортеже отношения.

Важно иметь в виду, что каждый атрибут может быть определен только на одном домене, однако разные атрибуты могут быть определены на одном и том же домене. Напр., домен дата может определять несколько атрибутов: дата рождения, дата окончания школы и т.д.

Число атрибутов в отношении называют *степенью* (или -*арностью*) отношения. Мощность множества кортежей отношения называют *мощностью* отношения.

Кортежи отношения обладают следующими свойствами:

- Каждый кортеж содержит точно одно значение (соответствующего типа) для каждого из своих атрибутов.
- Компоненты кортежа не упорядочены. Это свойство следует из того, что понятие кортежа определено на основании множества компонентов, а в математике элементы множества не упорядочены.
- Каждое подмножество кортежа представляет собой кортеж (а каждое подмножество заголовка является заголовком).

Пример 1. Отношение "Сотрудники" задано на доменах: "Номер_сотрудника", "Фамилия", "Зарплата", "Номер_отдела".

Заголовок отношения Сотрудники (Номер_сотрудника, Фамилия, Зарплата, Номер_отдела)

Представление отношения в виде таблицы приведено на рис. 3.1.

Номер_сотрудника	Фамилия	Зарплата	Номер_отдела
1	Иванов	1000	1
2	Петров	2000	2
3	Сидоров	3000	1

Рис. 3.1. Табличная форма представления отношения Сотрудник

Определение 3. **Реляционной БД** называется набор отношений.

Определение 4. **Схемой БД** данных называется набор схем отношений, входящих в БД и связи между отношениями.

Хотя любое отношение можно изобразить в виде таблицы, нужно понимать, что отношения не являются таблицами. Это близкие, но не совпадающие понятия. Термины, которыми оперирует реляционная модель данных, имеют соответствующие "табличные" синонимы (табл. 3.1).

Свойства табличного представления отношений

1. В отношении нет одинаковых кортежей.
2. Кортежи не упорядочены, т.к. отношения - множество, а множество не упорядочено.
3. Атрибуты не упорядочены. Т.к. каждый атрибут имеет уникальное имя в пределах отношения, то порядок атрибутов не имеет значения.
4. Все значения атрибутов атомарны, т.е. неделимы с точки зрения их обработки

В своей книге К.Дж. Дейт [1] обратил внимание на важные аспекты, отражающие смысл понятия отношения:

- в определенном отношении заголовок отношения *R* представляет собой определенный *предикат* (под *предикатом* подразумевается просто функция с истинностными значениями, которая, как и все функции, имеет ряд *формальных параметров).*

- каждая строка в теле отношения *R* представляет собой определенное *истинное высказывание***,** полученное из предиката путем подстановки определенных значений *фактических параметров* соответствующего типа вместо *формальных параметров* этого предиката.

Подобный взгляд на отношение имеет большое значение в плане реализации языковых средств работы с БД.

Таблица 3.1

Реляционный термин	**Соответствующий "табличный" термин**
База данных	Набор таблиц
Схема базы данных	Набор заголовков таблиц
Отношение	Таблица
Заголовок отношения	Заголовок таблицы
Тело отношения	Тело таблицы
Атрибут отношения	Наименование столбца таблицы
Кортеж отношения	Строка таблицы
Степень (-арность) отношения	Количество столбцов таблицы
Мощность отношения	Количество строк таблицы
Домены и типы данных	Типы данные в ячейках таблицы

3.2. Реляционная алгебра

Реляционная алгебра – это набор операций, которые выполняются над отношениями, результатом их выполнения также являются отношения. Первая версия этой алгебры была определена Коддом в статье « Relational Completeness of Data Base Sublanguages».

А*лгебра Кодда* состоит из восьми операций, которые делятся на два класса – теоретико-множественные операции и специальные реляционные операции. В состав теоретико-множественных операций входят операции:

- *объединения отношений*;
- *пересечения отношений*;
- *разность отношений*;
- *декартово произведения отношений*.

Специальные реляционные операции включают:

- *выбор*;
- *проекция отношения*;
- *соединение отношений*;
- *деление отношений*.

Геометрическая интерпретация операций реляционной алгебры приведена на рис. 3.2.

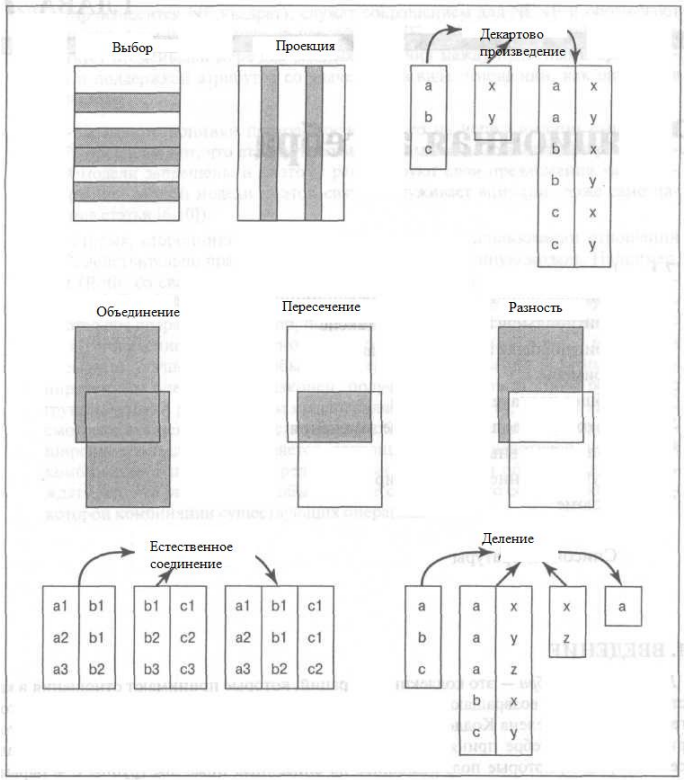

Рис. 3.2 Геометрическая интерпретация операций реляционной алгебры

Дж. Дейт предложил расширенный набор операций реляционной алгебры, включив некоторые дополнительные операции, например, *операция присваивания*, позволяющая сохранить в базе данных результаты вычисления алгебраических выражений, и *операция переименования атрибутов*, дающая возможность корректно сформировать заголовок (схему) результирующего отношения.

Следует отметить, что теоретико-множественные операции объединения, пересечения и разности применимы к отношениям, имеющим одинаковую схему.

3.3. Общая интерпретация реляционных операций

- 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 *выбора* (Select) отношения по некоторому условию является отношение, включающее кортежи отношения-операнда, удовлетворяющее заданному 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 работы с полями таблицы.

Данный набор операций реляционной алгебры обладает свойствами замкнутости и полноты.

При использовании реляционной алгебры для того, чтобы получить требуемый результат обработки таблиц, необходимо сформулировать последовательность операций для выполнения запроса. Однако существует другой подход к реализации обработки данных в отношениях. Он основан на использовании реляционного исчисления, который позволяет определить лишь характеристики результирующего отношения. В этом случае не надо задавать способ его формирования. При этом система сама

решает, какие операции и в каком порядке нужно выполнить над исходными отношениями, чтобы получить требуемый результат. На самом деле эти два механизма эквивалентны и существуют не очень сложные правила преобразования одного формализма в другой.

Реляционное исчисление является прикладной ветвью формального механизма исчисления предикатов первого порядка. Базисными понятиями исчисления являются понятие переменной с определенной для нее областью допустимых значений и понятие правильно построенной формулы, опирающейся на переменные, предикаты и кванторы. Применение названных подходов к обработке отношений привело к наличию разных языков работы с отношениями. В настоящее время стандартным подходом к реализации языков работы с отношениями является подход, основанных на реляционном исчислении. Именно он положен в основу стандарта на языки реляционных СУБД – SQL.

Контрольные вопросы

1. Дайте определение отношения.
2. Назовите основные элементы табличного представления отношения.
3. Дайте определение понятия атрибут отношения.
4. Что такое мощность отношения?
5. Назовите свойства табличного представления отношения
6. Перечислите операции реляционной алгебры (алгебры Кодда)
7. Приведите геометрическую интерпретацию специальных операций реляционной алгебры
8. Назовите свойства операций реляционной алгебры
9. В чем особенность применения теоретико-множественных операций реляционной алгебры?

ЛЕКЦИЯ 4. МЕТОДЫ ПРОЕКТИРОВАНИЕ РЕЛЯЦИОННОЙ МОДЕЛИ ДАННЫХ

Функциональные зависимости. Проектирование логической модели БД. Отображение концептуальной модели предметной области в реляционную модель БД. Нормальные формы, алгоритмы нормализации. Многозначные зависимости. CASE-средства проектирования моделей БД.

4.1. Аномалии реляционной модели БД

Проектирование логической модели БД должно решать задачи минимизации дублирования данных и упрощения процедур их обработки и обновления. При неправильно спроектированной схеме БД могут возникнуть аномалии модификации данных. Они обусловлены отсутствием средств явного представления типов множественных связей между объектами предметной области и неразвитостью средств описания ограничений целостности на уровне модели данных. Не все РБД обладают одинаковыми свойствами. В случае неправильного её проектирования реляционная модель БД (РМБД) может обладать аномалиями, которые значительно ухудшают характеристики её работы. Аномалии связаны с проблемами реализации операций изменения состояния БД. Различают следующие виды аномалий:

- Аномалии обновления.
- Аномалии добавления.
- Аномалии удаления.

Наличие аномалий связано со схемой отношения и наличием нежелательных функциональных зависимостей в схеме отношения.

Основной задачей логического этапа проектирования является РБД отображение объектов предметной области в объекты используемой модели данных. Такое отображение должно адекватно отображать семантику предметной области и быть наилучшим (эффективным, удобным и т.д.). С точки зрения выбранной СУБД задача логического проектирования реляционной базы данных состоит в обоснованном принятии решений о том:

- из каких отношений должна состоять база данных;
- какие атрибуты должны быть у этих отношений;
- какие ключевые атрибуты должны быть определены для каждого отношения;
- какие ограничения должны быть наложены на атрибуты и отношения базы данных, чтобы обеспечить ее целостность.

Требования к выбранному набору отношений и составу их атрибутов должны удовлетворять следующим условиям:

- отношения должны отличаться минимальной избыточностью атрибутов;
- выбранные для отношения первичные ключи должны быть минимальными;
- отношение должно находиться в 3НФ или НФБК;
- выбор отношений и атрибутов должен обеспечивать минимальное дублирование данных

Классический подход к проектированию РМБД основан на последовательном приближении схемы БД к удовлетворительному набору схем отношений. Основой этого процесса является представление предметной области в виде канонической схемы, которая представляет собой одно или несколько отношений. На каждом шаге проектирования производится изменение набора схем отношений, каждое из которых обладает лучшими свойствами. Сам процесс проектирования представляет собой *процесс нормализации* схем отношений, причем каждая следующая нормальная форма обладает свойствами, в некотором смысле, лучшими, чем предыдущая. Нормализация представляет собой декомпозицию отношения, находящегося в предыдущей нормальной форме, на два или более отношений, которые удовлетворяют требованиям следующей нормальной формы. В настоящее время. В связи с тем, что предметные области применения БД становятся очень сложными, используется подход, основанный на преобразовании ER-модели в РМБД, а затем полученная модель дорабатывается с учетом требований 3-й нормальной формы.

4.2. Получение реляционной схемы из ER-модели

Шаг 1. Каждая простая сущность превращается в таблицу. Простая сущность – сущность, не являющаяся подтипом и не имеющая подтипов. Имя сущности становится именем таблицы.

Шаг 2. Каждый атрибут становится возможным столбцом с тем же именем; может выбираться более точный формат. Столбцы, соответствующие необязательным атрибутам, могут содержать неопределенные значения; столбцы, соответствующие обязательным атрибутам, – не могут.

Шаг 3. Компоненты уникального идентификатора сущности превращаются в первичный ключ таблицы. Если имеется несколько возможных уникальных идентификатора, выбирается наиболее используемый. Если в состав уникального идентификатора входят связи, к числу столбцов первичного ключа добавляется копия уникального идентификатора сущности, находящейся на дальнем конце связи (этот процесс может продолжаться рекурсивно). Для именования этих столбцов используются имена концов связей и/или имена сущностей.

Шаг 4. Связи многие-к-одному (и один-к-одному) становятся внешними ключами. Т.е. делается копия уникального идентификатора с конца связи "один", и соответствующие столбцы составляют внешний ключ. Необязательные связи

соответствуют столбцам, допускающим неопределенные значения; обязательные связи – столбцам, не допускающим неопределенные значения.

Шаг 5. Индексы создаются для первичного ключа (уникальный индекс), внешних ключей и тех атрибутов, на которых предполагается в основном базировать запросы.

Шаг 6. Если в концептуальной схеме присутствовали подтипы, то возможны два способа:

- все подтипы в одной таблице (а)
- для каждого подтипа – отдельная таблица (б)

При применении способа (а) таблица создается для наиболее внешнего супертипа, а для подтипов могут создаваться представления. В таблицу добавляется по крайней мере один столбец, содержащий код ТИПА; он становится частью первичного ключа.

При использовании метода (б) для каждого подтипа первого уровня (для более нижних - представления) супертип воссоздается с помощью представления UNION (из всех таблиц подтипов выбираются общие столбцы - столбцы супертипа).

Сравнение рассмотренных способов реализации подтипов приведены в табл. 4.1.

Шаг 7. Имеется два способа работы при наличии исключающих связей:

- общий домен (а)
- явные внешние ключи (б)

Если остающиеся внешние ключи все в одном домене, т.е. имеют общий формат (способ (а)), то создаются два столбца: идентификатор связи и идентификатор сущности. Столбец идентификатора связи используется для различения связей, покрываемых дугой исключения. Столбец идентификатора сущности используется для хранения значений уникального идентификатора сущности на дальнем конце соответствующей связи. Если результирующие внешние ключи не относятся к одному домену, то для каждой связи, покрываемой дугой исключения, создаются явные столбцы внешних ключей; все эти столбцы могут содержать неопределенные значения.

Преимущества и недостатки этих способов реализации исключающих связей приведены в табл. 4.2.

Таблица 4.1.

Все подтипы в одной таблице	**Для каждого подтипа отдельная таблица**
Преимущества	
Все атрибуты хранятся вместе Легкий доступ к супертипу и подтипам Требуется меньше таблиц	Более ясны правила подтипов Программы работают только с нужными таблицами
Недостатки	
Требуется дополнительная логика работы с разными наборами столбцов и разными ограничениями. Потенциальное узкое место (в связи с блокировками) Столбцы подтипов должны быть необязательными В некоторых СУБД для хранения неопределенных значений требуется дополнительная память	Слишком много таблиц Смущающие столбцы в представлении UNION Потенциальная потеря производительности при работе через UNION Над супертипом невозможны модификации

Таблица 4.2.

Общий домен	**Явные внешние ключи**
Преимущества	
Нужно только два столбца	Условия соединения задаются явно
Недостатки	
Оба дополнительных атрибута должны использоваться в соединениях	Слишком много столбцов

4.3. Нормальные формы отношения

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 атомарны. В теории РБД обычно выделяется следующие нормальные формы:

- *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.

Ограничения нормальной формы связаны с понятием *функциональной зависимости* атрибутов. Дадим определение функциональной зависимости.

Пусть *X* и *Y* – атрибуты некоторого отношения *R*. Если в любой момент времени каждому значению *X* соответствует единственное значение *Y*, то *Y функционально зависит* от *X (X→Y)*.

Атрибут *Y* функционально полно зависимости от составного атрибута *X*, если он функционально зависит от *X* и не зависит функционально от любого подмножества атрибута *X*.

Теперь введем определения НФ.

Определение 3. Таблица находится в первой нормальной форме (1НФ) тогда и только тогда, когда каждое поле отношение содержит атомарное значение.

Определение 4. Отношение находится во второй нормальной форме (2НФ), если оно находится в 1НФ и каждый неключевой атрибут функционально полно зависит от ключа.

Определение 5. Отношение находится в третьей нормальной форме (3НФ), если оно находится во 2НФ и каждый неключевой атрибут нетранзитивно зависит от первичного ключа.

Определение 6. Отношение находится в *нормальной форме Бойса-Кодда* (*НФБК*), тогда и только тогда, когда любая функциональная зависимость между его атрибутами сводится к полной функциональной зависимости от *вероятностного* ключа.

Для получения качественной РБД в большинстве случаев достаточно приведение схемы её отношений в 3НФ или НФБК. Для достижения этого используют процесс нормализации, т.е. приведения отношения в требуемую НФ. Процесс нормализации схемы отношения, входящего в состав РБД, выполняется путём её декомпозиции, разбиения отношения на отношения с более простой схемой. *Декомпозицией* схемы отношения R называется замена её совокупностью схем отношений R_i таких, что

$$\boldsymbol{R} = \bigcup_{i} \boldsymbol{R_i}$$

При этом не требуется, чтобы отношения R_i были непересекающимися.

Декомпозиция должна удовлетворять двум основным свойствам:

- *соединения без потерь*
- *сохранение зависимостей.*

При определении декомпозиции, удовлетворяющей указанным свйоствам, используется теорема Хита.

Теорема Хита. Пусть *R(A, B, C)* является отношением, где *A, B* и *C* – атрибуты этого отношения. Если *B* функционально зависит от *A*, то *R* равно соединению его проекций {A, B}и {A, C}.

При определении четвертой нормальной формы (4НФ) используется понятие многозначной зависимости.

Определение 7. Многозначная зависимость. Пусть *A, B* и *C* – подмножества атрибутов *R* . *B* → *A*, тогда и только тогда, когда множество значений *B*, соответствующее заданной паре *(A,C)* в *R* , зависит от *A*, но не зависит от *C*.

Теорема Фейгина (более строгая форма теоремы Хита). Пусть *A,B,C* – множества атрибутов в *R*. *R* будет равна соединению ее проекций *{A,B}* и *{A,C}* тогда и только тогда, когда для *R* выполняется *B* → *A,*

Определение 8. Отношение R находится в 4НФ, если она находится в НФБК и все многозначные зависимости фактически представляют собой функциональные зависимости от ее ключей.

Определение 9. Отношение *R* удовлетворяет зависимости соединения тогда и только тогда, когда любое допустимое значение R эквивалентно соединению ее проекций по подмножествам ее атрибутов.

Определение 10. Отношение *R* находится в 5НФ (проекционно-соединительной НФ) тогда и только тогда, когда каждая нетривиальная зависимость соединения (т.е. не совпадающая с *R*) подразумевается ее потенциальными ключами.

Процесс нормализации включает следующие шаги:

1. Отношение в 1НФ разбить на проекции, которые позволяют исключить все функциональные зависимости, не являющиеся неприводимыми. Получаем набор отношений в 2НФ.

2. Отношений в 2НФ разбиваем на проекции, в которых нет транзитивных зависимостей. Получаем набор отношений в 3НФ.

3. Отношения в 3НФ разбиваем на проекции, в которых нет отношений с детерминантами, не являющимися потенциальными ключами. Получаем набор отношений в НФБК.

4. Отношения НФБК разбиваем на проекции, исключающие многозначные зависимости , которые не являются функциональными. Получаем набор отношений в 4НФ.

5. Исключаем зависимости соединения для 4НФ.

4.4. Ограниченность реляционной модели при проектировании баз данных

- Модель не обеспечивает достаточных средств для представления семантики предметной области. Это относится к проблеме представления ограничений целостности.

- Во многих прикладных областях трудно моделировать предметную область на основе двумерных таблиц.

- Хотя весь процесс проектирования происходит на основе учета зависимостей, реляционная модель не предоставляет какие-либо формализованные средства для представления этих зависимостей и их анализа.

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

4.5. CASE-средства проектирования

4.5.1. Назначение CASE-средств

К CASE-средствам (Computer Aided Software Engineering) относят любое программное средство, которое автоматизирует ту или иную совокупность процессов жизненного цикла программного обеспечения. CASE-средства обладают следующими общими свойствами:

- наличие мощных графических средств для описания и документирования моделей, которые обеспечивают удобный интерфейс с разработчиком и развивают его творческие возможности, позволяя сосредоточиться на смысловых аспектах модели;
- поддержка интеграции отдельных компонент CASE-средств, которая позволяет управлять процессом разработки БД;
- использование специальным образом организованного хранилища проектных метаданных (репозитория).

В структуре е CASE-средства, как правило, выделяют следующие компоненты:

- репозиторий, являющийся основой CASE-средства. Он обеспечивает хранение версий проекта и его отдельных компонентов, синхронизацию поступления информации от различных разработчиков при групповой разработке, контроль метаданных на полноту и непротиворечивость;
- графические средства анализа и проектирования, обеспечивающие создание и редактирование иерархически связанных диаграмм (DFD, ER-диаграмма и др.);
- средства разработки приложений, включая языки 4GL и генераторы кодов;
- средства конфигурационного управления;
- средства документирования;
- средства тестирования;
- средства управления проектом;
- средства реинжиниринга.

Классификация CASE-средств может быть проведена на основании различных признаков:

- по ориентации на процессы жизненного цикла,

- по степени интегрированности
- по применяемым методологиям и моделям систем и БД;
- по степени интегрированности с СУБД;
- по доступным платформам.

Классификация по ориентации на модели жизненного цикла включает следующие основные классы:

- средства анализа (Upper CASE), предназначенные для построения и анализа моделей предметной области (Design/IDEF (Meta Software), BPwin (Logic Works));
- средства анализа и проектирования (Middle CASE), поддерживающие наиболее распространенные методологии проектирования и использующиеся для создания проектных спецификаций (Vantage Team Builder (Cayenne), Designer/2000 (ORACLE), Silverrun (CSA), PRO-IV (McDonnell Douglas), CASE.Аналитик (МакроПроджект)). Выходом таких средств являются спецификации компонентов и интерфейсов системы, архитектуры системы, алгоритмов и структур данных;
- средства проектирования баз данных, обеспечивающие моделирование данных и генерацию схем баз данных (как правило, на языке SQL) для наиболее распространенных СУБД. К ним относятся ERwin (Logic Works), S-Designor (SDP) и DataBase Designer (ORACLE). Средства проектирования баз данных имеются также в составе CASE-средств Vantage Team Builder, Designer/2000, Silverrun и PRO-IV;
- средства разработки приложений. К ним относятся средства 4GL (Uniface (Compuware), JAM (JYACC), PowerBuilder (Sybase), Developer/2000 (ORACLE), New Era (Informix), SQL Windows (Gupta), Delphi (Borland) и др.) и генераторы кодов, входящие в состав Vantage Team Builder, PRO-IV и частично - в Silverrun;
- средства реинжиниринга, обеспечивающие анализ программных кодов и схем баз данных и формирование на их основе различных моделей и проектных спецификаций. Средства анализа схем БД и формирования ER-диаграмм входят в состав Vantage Team Builder, PRO-IV, Silverrun, Designer/2000, ERwin и S-Designor. В области анализа программных кодов наибольшее распространение получают объектно-ориентированные CASE-средства, обеспечивающие реинжиниринг программ на языке Си++ (Rational Rose (Rational Software), Object Team (Cayenne));
- средства планирования и управления проектом (SE Companion, Microsoft Project и др.);
- средства конфигурационного управления (PVCS (Intersolv));
- средства тестирования (Quality Works (Segue Software));

- средства документирования (SoDA (Rational Software)).

Ниболее развитые CASE-средства :

- Vantage Team Builder (Westmount I-CASE);
- Designer/2000;
- Silverrun;
- CA Erwin Modeling Suite;
- S-Designor;
- CASE.Аналитик.

Ниже приводится краткая характеристика приведенных CASE-средств.

4.5.2. Designer/2000 (Oracle)

Designer/2000 (предыдущие версии продукта назывались Oracle*CASE представляет собой универсальное CASE-средство, позволяющее моделировать бизнес-процессы, создавать диаграммы потоков данных и функциональные модели. Средство проектирования данных и создания ER-диаграмм является лишь одной из составных частей этого довольно сложного продукта и предоставляет возможность сохранять созданные модели данных и описанные бизнес-правила в предназначенном для этого репозитарии.

Designer/2000, предназначенный для использования главным образом с Oracle 8 и более поздних версий продукта, поддерживает все особенности данной СУБД, включая объектные типы данных (CLOB, Arrays, вложенные таблицы и др.). Это CASE-средство позволяет создать определения ролей, сгенерировать триггеры, реализующие бизнес-логику, которая описана в моделях, используемых при генерации базы данных, а также генерировать объекты для распределенных базы данных. Кроме того, с помощью Designer/2000 можно создавать физические модели и осуществлять обратное проектирование и для других СУБД – Oracle RDB, DB2, Microsoft SQL Server, Sybase, ODBC-источников данных, а также осуществлять обратное проектирование на основании DDL-сценариев, если они соответствуют стандарту ANSI SQL.

Весьма привлекательной особенностью Designer/2000 является возможность генерации форм Oracle Developer/2000, проектов Visual Basic, классов C++, отчетов Oracle Reports, приложений для Oracle Web Application Server.

4.5.3. CA Erwin Modeling Suite

В состав CASE-средства CA Erwin Modeling Suite входит целая группа средств, обеспечивающих моделирование и анализ процессов и потоков данных (Process Modeler – ранее BPwin) Erwin Data Modeler (ранее ERwin), анализ полученных моделей и их сопровождение. Для проектирования моделей БД в этом продукте предназначен компонент Erwin Data Modeler.

Erwin Data Modeler не ориентирован на какую-то конкретную СУБД и поддерживает более 20 типов СУБД, включая СУБД всех ведущих производителей серверов баз данных (Oracle, Sybase, Microsoft, IBM, Informix), а также все популярные форматы настольных СУБД (включая dBase, Clipper, FoxPro, Access, Paradox). Это CASE-средство остается одним из самых популярных в мире продуктов этого класса благодаря поддержке большого количества платформ, простоте интерфейса и, что немаловажно, поддержке специфических особенностей организации физической памяти наиболее популярных серверных СУБД. Оно обладает встроенным макроязыком для написания в процессе логического проектирования не зависящих от СУБД шаблонов серверного кода, а также готовыми шаблонами для генерации триггеров, реализующих стандартные действия (например, каскадное удаление). При необходимости можно создавать и свои шаблоны триггеров, используя этот язык, причем каждая таблица может иметь свой набор шаблонов. При создании физической модели шаблоны преобразуются в код на процедурном расширении SQL того сервера, для которого создается физическая модель. Логическая и физическая модели Erwin Data Modeler хранятся в одном файле. Erwin Data Modeler поддерживает обмен моделями с репозитарием Designer/2000 и Microsoft Repository, а также генерацию клиентских приложений для Visual Basic и PowerBuilder.

Недавно компанией Computer Associates был выпущен новый продукт – ERwin Examiner, представляющий собой инструмент проверки баз данных и DDL-скриптов с целью выявления ошибок проектирования данных, сказывающихся на целостности данных и производительности сервера, таких, например, как ошибки нормализации, противоречивые ключи и т.д. В результате проверки ERwin Examiner предлагает способы устранения найденных ошибок, генерируя соответствующие DDL-скрипты.

4.5.4. PowerDesigner (Sybase)

PowerDesigner (бывший S-Designor, принадлежавший компании PowerSoft) представляет собой инструмент, в состав которого входят средство создания концептуальных (то есть логических) моделей, средство создания физических моделей и средство объектно-ориентированного моделирования, используемое при генерации клиентских приложений. Средство создания физических моделей представляет собой отдельный продукт – PowerDesigner PhysicalArchitect. В состав продукта PowerDesigner DataArchitect входят средства создания концептуальных и физических моделей, в состав PowerDesigner Developer – средства объектно-ориентированного моделирования и создания физических моделей, а в состав PowerDesigner ObjectArchitect – все три средства.

Физические и концептуальные модели в PowerDesigner DataArchitect хранятся в разных файлах, однако возможна генерация как физической модели на основе модели концептуальной, так и наоборот.

Помимо серверных СУБД производства Sybase (Adaptive Server Enterprise 12.0, Sybase SQL Anywhere) PowerDesigner DataArchitect способен работать с любыми ODBC-источниками. Как и ERwin, он поддерживает генерацию триггеров серверных СУБД, осуществляющих стандартную обработку событий, связанных с нарушениями ссылочной целостности.

PowerDesigner Developer и PowerDesigner ObjectArchitect могут генерировать код клиентских приложений для PowerBuilder, а также классы Java и компоненты JavaBeans. Возможно и обратное проектирование диаграмм классов из исходных текстов Java, байт-кодов и архивов Java. Поддерживается также генерация кода Web-приложений и объектов для Sybase Enterprise Application Server на основе физической модели.

PowerDesigner DataArchitect может импортировать логические и физические модели ERwin. Это CASE-средство позволяет хранить свои модели данных в коллективно разделяемом репозитарии, которое управляется с помощью средства PowerDesigner MetaWorks.

4.5.5. ER/Studio (Embarcadero Technologies)

ER/Studio является менее популярным в нашей стране, чем Erwin Data Modeler и PowerDesigner DataArchitect. По своим возможностям этот продукт сходен с Erwin Data Modeler. Список поддерживаемых им СУБД у этого продукта достаточно широк и включает все наиболее популярные серверные и настольные СУБД. ER/Studio поддерживает написание макросов на SAX Basic (клон Visual Basic for Applications). Этот язык позволяет создавать макросы для выполнения однотипных операций, например добавления стандартных полей к вновь создаваемым сущностям. С помощью этого же языка можно генерировать стандартные триггеры и хранимые процедуры для вставки, удаления, изменения записей. Однако, в отличие от ERwin, ER/Studio не позволяет добавить к каждой таблице свои шаблоны триггеров или просмотреть код конкретного триггера в процессе разработки модели — чтобы получить код одного триггера, нужно сгенерировать скрипт для всей модели.

Модели ER/Studio можно сохранить не только в виде DDL-скрипта, но и в формате XML. Можно также создать репозитарий для их хранения в любой серверной СУБД. ER/Studio может импортировать модели ERwin, но при импорте теряются связи шаблонов серверного кода с конкретными таблицами, и не все макросы ERwin корректно преобразуются в макросы SAX Basic. ER/Studio позволяет сгенерировать Java-классы для клиентских приложений.

Следует отметить, что ER/Studio является COM-сервером, что позволяет использовать его в других приложениях, предоставляя им возможность просмотра и редактирования моделей данных, а также создавать другие решения на его основе.

4.5.6. Visible Analyst (Visible Systems Corporation)

Visible Analyst — весьма популярный продукт компании Visible Systems Corporation. Широко известны также ранее производимые этой компанией CASE-средства EasyER и EasyCASE – предшественники Visible Analyst. Этот продукт выпускается в трех редакциях: Visible Analyst DB Engineer, который включает средства проектирования данных, Visible Analyst Standard, который кроме проектирования данных позволяет осуществлять структурное моделирование, и Visible Analyst Corporate, который помимо указанных выше возможностей позволяет осуществлять также объектно-ориентированное моделирование.

Visible Analyst поддерживает довольно широкий спектр СУБД с точки зрения генерации серверного кода, включая Oracle 7, Sybase SQL Server (System 10 и 4.x); Informix, DB2, Ingres. Для Informix и DB2 указанный продукт позволяет генерировать DDL-скрипты, учитывающие специфические особенности организации физической памяти наиболее популярных серверных СУБД. Они позволяют осуществлять управление табличным пространством, размером экстентов, режимами блокировки данных, степенью заполнения данными (fill factor), а также создавать кластеризованные индексы и генерировать триггеры для выполнения стандартных операций. Из этих же СУБД можно производить непосредственно обратное проектирование. Помимо этих двух СУБД обратное проектирование можно производить также из DDL-скриптов, сгенерированных для других СУБД, а также на основе кода COBOL.

Модели Visible Analyst можно сохранять в многопользовательском репозитарии, созданном в одной из серверных СУБД.

Кроме того, Visible Analyst позволяет на основе созданных моделей генерировать код для Visual Basic, C++ и COBOL.

4.5.7. Visio Enterprise (Microsoft)

Продукт под названием Visio, приобретенный в январе 2000 года корпорацией Microsoft вместе с его разработчиком – компанией Visio Corporation, позиционировался на рынке как одно из самых популярных средств создания схем и диаграмм. Visio Enterprise позволяет производить прямое и обратное проектирование данных, преобразовывать логическую модель в физическую. Этим средством поддерживаются все ODBC- и OLE DB-источники данных. С его помощью можно создавать триггеры для стандартной обработки нарушений ссылочной целостности в случае, если DDL-скрипт создается для Microsoft SQL Server, и серверные ограничения, если скрипт создается для другой СУБД. Отметим, что Visio при генерации скриптов позволяет указывать параметры организации физической памяти Oracle, Informix, Microsoft SQL Server, DB2 и некоторых других СУБД.

Отметим, что помимо средств проектирования данных Visio включает средства объектно-ориентированного моделирования и генерации кода приложений Visual Basic 6, а также классов C++ и Java. Модели Visio можно сохранять в Microsoft Repository.

Visio, в отличие от специализированных средств проектирования данных, не обладает скриптовым языком, позволяющим создавать серверный код, не связанный с конкретной СУБД. При использовании этого продукта такой код нужно создавать на этапе физического проектирования в уже созданном скрипте. Однако справедливости ради заметим, что и стоимость Visio Enterprise по сравнению с ERwin или PowerDesigner DataArchitect невысока, тем более что Visio в целом представляет собой продукт более широкого назначения, нежели другие рассмотренные выше средства проектирования данных. К тому же этот продукт является сервером автоматизации, обладает весьма обширной объектной моделью и встроенным средством разработки – Visual Basic for Applications, что позволяет, в частности, создавать на его базе разнообразные решения, в том числе и автоматизировать разработку моделей данных.

Контрольные вопросы

1. Что такое аномалии БД?
2. Назовите виды аномалий БД.
3. Что является источником аномалий БД?
4. Назовите шаги получения реляционной схемы БД из ER-модели.
5. Дайте определение 1НФ
6. Определите сущность функциональной зависимости.
7. Дайте определение 3НФ.
8. Приведите определение нормальной формы Бойса-Кодда
9. Определите понятие декомпозиционной схемы отношения.
10. Сформулируйте теорему Хита.
11. Для чего используются CASE-средства?
12. Перечислите компоненты CASE-средства.
13. Дайте характеристику CASE-средства Erwin Modeling Suite.
14. Охарактеризуйте CASE-средство PowerDesigner.

ЛЕКЦИЯ 5. ФИЗИЧЕСКИЙ УРОВЕНЬ ПРЕДСТАВЛЕНИЯ

Методы внутримашинного представления данных. Методы физической организации БД. Способы индексации..

Физические модели баз данных определяют способы размещения данных в среде хранения и способы доступа к этим данным, которые поддерживаются на физическом уровне. Исторически первыми системами хранения и доступа были файловые структуры и системы управления файлами, которые фактически являлись частью операционных систем. СУБД создавала над этими файловыми моделями спою надстройку, которая позволяла организовать всю совокупность файлов таким образом, чтобы она работала как единое целое и получала централизованное управление от СУБД. Однако непосредственный доступ осуществлялся на уровне файловых команд, которые СУБД использовала при манипулировании всеми файлами, составляющими хранимые данные одной или нескольких баз данных. Однако механизмы буферизации и управления файловыми структурами не приспособлены для решения задач собственно СУБД, эти механизмы разрабатывались просто для традиционной обработки файлов, и с ростом объемов хранимых данных они стали неэффективными для использования СУБД. Тогда постепенно произошел переход от базовых файловых структур к непосредственному управлению размещением данных на внешних носителях самой СУБД. И пространство внешней памяти уже выходило из-под владения систем управления файлами и управлялось непосредственно СУБД. При этом механизмы, применяемые в файловых системах, перешли во многом и в новые системы организации данных во внешней памяти, называемые чаще страничными системами хранения информации. В каждой СУБД по-разному организованы хранение и доступ к данным, однако существуют некоторые файловые структуры, которые имеют общепринятые способы организации и широко применяются практически во всех СУБД.

5.1. Методы внутримашинного представления

В БД файлы и файловые структуры, которые используются для хранения информации во внешней памяти, можно классифицировать следующим образом (рис. 5.1).

Рис. 5.1. Классификация файлов, используемых в системах баз данных

С точки зрения пользователя, *файлом* называется поименованная линейная последовательность записей, расположенных на внешних носителях. На рис. 5.2 представлена такая условная последовательность записей. Так как файл – это линейная последовательность записей, то всегда в файле можно определить текущую запись, предшествующую ей и следующую за ней. Всегда существует понятие первой и последней записи файла. В соответствии с методами управления доступом различают

устройства внешней памяти с *произвольной адресацией* (магнитные и оптические диски) и устройства *с последовательной адресацией* (магнитофоны, стримеры).

На устройствах с произвольной адресацией теоретически возможна установка головок чтения-записи в произвольное место мгновенно. Практически существует время позиционирования головки, которое весьма мало по сравнению со временем считывания-записи. В устройствах с последовательным доступом для получения доступа к некоторому элементу требуется «перемотать (пройти)» все предшествующие ему элементы информации. На устройствах с последовательным доступом вся память рассматривается как линейная последовательность информационных элементов (рис. 5.3).

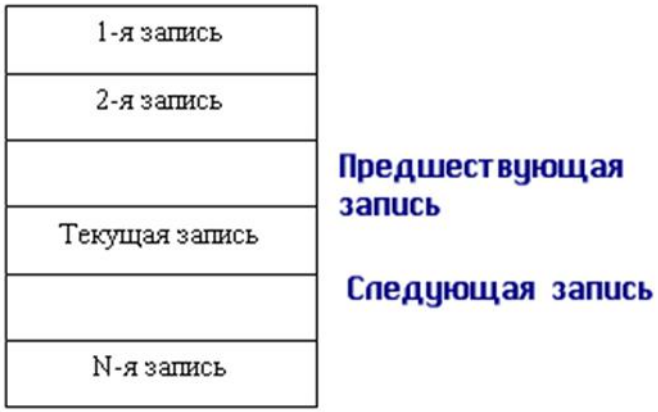

Рис. 5.2. Абстрактное представление файла

Рис. 5.3. Модель хранения информации на устройстве последовательного доступа

Файлы с постоянной длиной записи, расположенные на устройствах прямого доступа, могут быть реализованы как *файлы прямого доступа.* В этих файлах физический адрес расположения нужной записи может быть вычислен по номеру записи (*NZ*).

Каждая файловая система, как правило, поддерживает некоторую иерархическую файловую структуру, которая включает чаще всего неограниченное количество уровней иерархии в представлении внешней памяти (рис. 5.4).

Для каждого файла в системе хранится следующая информация:

- имя файла;
- тип файла (например, расширение или другие характеристики);
- размер записи;
- количество занятых физических блоков;
- базовый начальный адрес;
- ссылка на сегмент расширения;
- способ доступа (код защиты).

Рис. 5.4. Иерархическая организация файловой структуры хранения

Для файлов с постоянной длиной записи адрес размещения записи с номером *K* может быть вычислен по формуле:

*BA + (K - 1) * LZ* + 1, где *BA* –базовый адрес, *LZ* – длина записи.

На устройствах последовательного доступа могут быть организованы файлы только последовательного доступа. Файлы с переменной длиной записи всегда являются файлами последовательного доступа. Они могут быть организованы двумя способами:

- Конец записи отличается специальным маркером.

Запись 1	X	Запись 2	X	ЗаписьЗ	X

- В начале каждой записи записывается ее длина.

LZ1	Запись1	LZ2	Запись2	LZ3	ЗаписьЗ

Здесь *LZN* — длина *N*-й записи.

5.1.1. Файлы прямого доступа

Файлы с прямым доступом обеспечивают наиболее быстрый способ доступа. Мы не всегда можем хранить информацию в виде файлов прямого доступа, но главное – это то, что доступ по номеру записи в базах данных весьма неэффективен. Чаще всего в базах данных необходим поиск по первичному или возможному ключам, иногда необходима выборка по внешним ключам, но во всех этих случаях мы знаем значение ключа, но не знаем номера записи, который соответствует этому ключу.

При организации файлов прямого доступа в некоторых очень редких случаях возможно построение функции, которая по значению ключа однозначно вычисляет адрес (номер записи файла).

$NZ = F(K)$, где NZ – номер записи, K – значение ключа, $F( )$ – функция.

В этом случае не удается построить взаимно-однозначную функцию, либо эта функция будет иметь множество незадействованных значений, которые соответствуют недопустимым значениям ключа. В подобных случаях применяют различные методы хеширования (рандомизации) и создают специальные хеш- функции.

Суть методов хеширования состоит в том, что мы берем значения ключа (или некоторые его характеристики) и используем его для начала поиска, то есть мы вычисляем некоторую хэш-функцию $h(k)$ и полученное значение берем в качестве адреса начала поиска. То есть мы не требуем полного взаимно-однозначного соответствия, но, с другой стороны, для повышения скорости мы ограничиваем время этого поиска (количество дополнительных шагов) для окончательного получения адреса. Таким образом, мы допускаем, что нескольким разным ключам может соответствовать одно значение хэш-функции (то есть один адрес). Подобные ситуации называются *коллизиями.* Значения ключей, которые имеют одно и то же значение хэш-функции, называются *синонимами* (рис. 5.5).

Рис. 5.5. Синонимы

Поэтому при использовании хеширования как метода доступа необходимо принять два независимых решения:

- выбрать хэш-функцию;
- выбрать метод разрешения коллизий.

Существует множество различных стратегий разрешения коллизий, но мы для примера рассмотрим две достаточно распространенные.

Стратегия разрешения коллизий с областью переполнения

Первая стратегия условно может быть названа стратегией с областью переполнения. При выборе этой стратегии область хранения разбивается на 2 части:

- основную область;
- область переполнения.

Для каждой новой записи вычисляется значение хэш-функции, которое определяет адрес ее расположения, и запись заносится в основную область в соответствии с полученным значением хэш-функции. Если вновь заносимая запись имеет значение функции хеширования такое же, которое использовала другая запись, уже имеющаяся в БД, то новая запись заносится в область переполнения на первое свободное место, а в записи-синониме, которая находится в основной области, делается ссылка на адрес вновь размещенной записи в области переполнения. Если же уже существует ссылка в записи-синониме, которая расположена в основной области, то тогда новая запись получает дополнительную информацию в виде ссылки и уже в таком виде заносится в область переполнения (рис. 5.6).

При этом цепочка синонимов не разрывается, но мы не просматриваем ее до конца, чтобы расположить новую запись в конце цепочки синонимов, а располагаем всегда новую запись на второе место в цепочке синонимов, что существенно сокращает время размещения новой записи. При таком алгоритме время размещения любой новой записи составляет не более двух обращений к диску, с учетом того, что номер первой свободной записи в области переполнения хранится в виде системной переменной.

Рассмотрим теперь механизмы поиска произвольной записи и удаления записи для этой стратегии хеширования.

При поиске записи также сначала вычисляется значение ее хэш-функции и считывается первая запись в цепочке синонимов, которая расположена в основной области. Если искомая запись не соответствует первой в цепочке синонимов, то далее поиск происходит перемещением по цепочке синонимов, пока не будет обнаружена требуемая запись. Скорость поиска зависит от длины цепочки синонимов, поэтому качество хэш-функции определяется максимальной длиной цепочки синонимов. Хорошим результатом может считаться наличие не более 10 синонимов в цепочке.

Рис. 5.6. Формирование области переполнения

При удалении произвольной записи сначала определяется ее место расположения. Если удаляемой является первая запись в цепочке синонимов, то после удаления на ее место в основной области заносится вторая (следующая) запись в цепочке синонимов, при этом все указатели (ссылки на синонимы) сохраняются.

Если же удаляемая запись находится в середине цепочки синонимов, то необходимо провести корректировку указателей: в записи, предшествующей удаляемой, в цепочке ставится указатель из удаляемой записи. Если это последняя запись в цепочке, то все равно механизм изменения указателей такой же, то есть в предшествующую запись заносится признак отсутствия следующей записи в цепочке, который ранее хранился в последней записи.

5.1.2. Организация стратегии свободного замещения

При этой стратегии файловое пространство не разделяется на области, но для каждой записи добавляется 2 указателя: указатель на предыдущую запись в цепочке синонимов и указатель на следующую запись в цепочке синонимов. Отсутствие соответствующей ссылки обозначается специальным символом, например нулем. Для каждой новой записи вычисляется значение хэш-функции, и если данный адрес свободен, то запись попадает на заданное место и становится первой в цепочке синонимов. Если адрес, соответствующий полученному значению хэш-функции, занят, то по наличию ссылок определяется, является ли запись, расположенная по указанному адресу, первой в цепочке синонимов. Если да, то новая запись располагается на первом свободном месте и для нее устанавливаются соответствующие ссылки: она становится второй в цепочке синонимов, на нее ссылается первая запись, а она ссылается на следующую, если таковая есть. Если запись, которая занимает требуемое место, не является первой записью в цепочке синонимов, то она перемещается на новое место. Механизм перемещения аналогичен занесению новой записи, которая уже имеет синоним, занесенный в файл. Для этой записи ищется первое свободное место и корректируются соответствующие ссылки: в записи, которая является предыдущей в цепочке синонимов для перемещаемой записи, заносится указатель на новое место перемещаемой записи, указатели же в самой перемещаемой записи остаются прежние.

После перемещения «незаконной» записи вновь вносимая запись занимает свое законное место и становится первой записью в новой цепочке синонимов. Механизмы удаления записей во многом аналогичны механизмам удаления в стратегии с областью переполнения. Однако еще раз кратко опишем их. Если удаляемая запись является первой записью в цепочке синонимов, то после удаления на ее место перемещается следующая (вторая) запись из цепочки синонимов и проводится соответствующая корректировка указателя третьей записи в цепочке синонимов, если таковая существует.

Если же удаляется запись, которая находится в середине цепочки синонимов, то производится только корректировка указателей: в предшествующей записи указатель на удаляемую запись заменяется указателем на следующую за удаляемой запись, а в записи, следующей за удаляемой, указатель на предыдущую запись заменяется на указатель на запись, предшествующую удаляемой.

5.2. Методы управления физической организацией БД

Для ускорения обработки данных разработчики СУБД реализуют свои способы построения физической модели. Для каждой СУБД способы построения физических моделей однозначно определено и пользователь СУБД не имеет возможности влиять на способ построения физической модели. Физическая организация современных баз данных является, в большинстве случаев, коммерческой тайной для большинства поставщиков коммерческих СУБД. Не существует никаких стандартов, поэтому в общем случае каждый поставщик создает свою уникальную структуру и пытается обосновать ее наилучшие качества по сравнению со своими конкурентами. Физическая организация является в настоящий момент наиболее динамичной частью СУБД.

К физической модели предъявляются следующие требования:

- высокая скорость доступа к данным;
- простота обновления данных.
- небольшой объем дополнительно используемой (вторичной) памяти.

При распределении дискового пространства рассматриваются (рис. 5.7) две схемы структуризации: физическая, которая определяет хранимые данные, и логическая, которая определяет некоторые логические структуры, связанные с концептуальной моделью данных

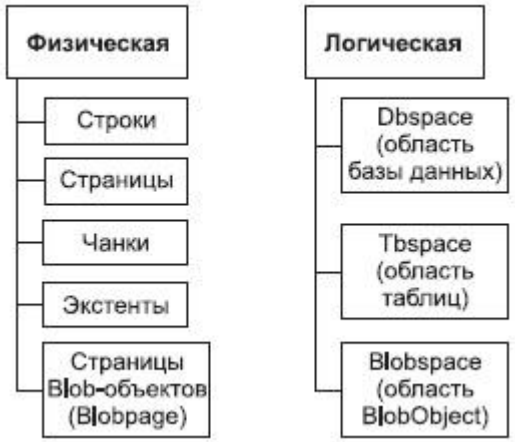

Рис. 5.7. Логическая и физическая схемы структуризации дискового пространства

Чанк (chank) – представляет собой часть диска, физическое пространство на диске, которое ассоциировано одному процессу (online процессу обработки данных).

Чанком может быть назначено неструктурированное устройство, часть этого устройства, блочно-ориентированное устройство или просто файл UNIX.

Чанк характеризуется маршрутным именем, смещением (от физического начала устройства до начальной точки на устройстве, которая используется как чанк), размером, заданным в Кбайтах или Мбайтах.

При использовании блочных устройств и файлов величина смещения считается равной нулю.

Логические единицы образуются совокупностью экстентов, то есть таблица моделируется совокупностью экстентов.

Экстент – это непрерывная область дисковой памяти.

Для моделирования каждой таблицы используется 2 типа экстентов: первый и последующие.

Первый экстент задается при создании нового объекта типа таблица, его размер задается при создании. EXTENTSIZE – размер первого экстента, NEXT SIZE — размер каждого следующего экстента.

Минимальный размер экстента в каждой системе свой, но в большинстве случаев он равен 4 страницам, максимальный – 2 Гбайтам.

Новый экстент создается после заполнения предыдущего и связывается с ним специальной ссылкой, которая располагается на последней странице экстента. В ряде систем экстенты называются сегментами, но фактически эти понятия эквиваленты.

При динамическом заполнении БД данными применяется специальный механизм адаптивного определения размера экстентов. Внутри экстента идет учет свободных станиц. Между экстентами, которые располагаются друг за другом без промежутков, производится своеобразная операция конкатенации, которая просто увеличивает размер первого экстента. При этом используется механизм удвоения размера экстента: если число выделяемых экстентов для процесса растет в пропорции, кратной 16, то размер экстента удваивается каждые 16 экстентов.

Например, если размер текущего экстента 16 Кбайт, то после заполнения 16 экстентов данного размера размер следующего будет увеличен до 32 Кбайт.

Совокупность экстентов моделирует логическую единицу – таблицу-отношение (tblspace).

Экстенты состоят из четырех типов страниц: страницы данных, страницы индексов, битовые страницы и страницы blob-объектов. Blob –это сокращение Binary Larg Object, и соответствует оно неструктурированным данным. В современных СУБД к этому типу относятся неструктурированные большие текстовые данные, картинки, просто наборы машинных кодов. СУБД важно знать, что этот объект надо хранить целиком, что размеры этих объектов от записи к записи могут отличаться, и этот размер в общем случае неограничен.

Основной единицей осуществления операций обмена (ввода-вывода) является страница данных. Все данные хранятся постранично. При табличном хранении данные на одной странице являются однородными, то есть станица может хранить только данные или только индексы.

Все страницы данных имеют одинаковую (рис. 5.8) структуру.

Слот – это 4-байтовое слово, 2 байта соответствуют смещению строки на странице и 2 байта — длина строки. Слоты характеризуют размещение строк данных на странице. На одной странице хранится не более 255 строк. В базе данных каждая строка имеет уникальный идентификатор в рамках всей базы данных, часто называемый RowID — номер строки, он имеет размер 4 байта и состоит из номера страницы и номера строки на странице. Под номер страницы отводится 3 байта, поэтому при такой идентификации возможна адресация к 16 777 215 страницам.

Рис. 5.8. Структура страницы

При упорядочении строк на страницах не происходит физического перемещения строк, все манипуляции происходят со слотами. При переполнении страниц создается специальный вид страниц, называемых страницами остатка. Строки, не уместившиеся на основной странице, связываются (линкуются) со своим продолжением на страницах остатка с помощью ссылок-указателей "вперед" (то есть на продолжение), которые содержат номер страницы и номер слота на странице.

Страницы индексов организованы в виде B-деревьев. Страницы blob предназначены для хранения слабоструктурированной информации, содержащей тексты большого объема, графическую информацию, двоичные коды. Эти данные рассматриваются как потоки байтов произвольного размера, в страницах данных делаются ссылки на эти страницы.

Битовые страницы служат для трассировки других типов страниц. В зависимости от трассируемых страниц битовые страницы строятся по 2-битовой или 4-битовой схеме. 4-битовые страницы служат для хранения сведений о столбцах типа Varchar, Byte, Text, для остальных типов данных используются 2-битовые страницы.

Битовая структура трассирует 32 страницы. Каждая битовая структура представлена двумя 4-байтными словами. Каждая i-я позиция описывает одну *i*-ю страницу. Сочетание разрядов в *i*-х позициях двух слов обозначает состояние данной страницы: ее тип и занятость.

При обработке данных СУБД организует специальные структуры в оперативной памяти, называемые разделяемой памятью, и специальные структуры во внешней памяти, называемые журналами транзакций. Разделяемая память служит для кэширования данных при работе с внешней памятью с целью сокращения времени доступа, кроме того, разделяемая память служит для эффективной поддержки режимов одновременной параллельной работы пользователей с базой данных.

Как правило, СУБД располагает своими собственными буферами оперативной памяти ЭВМ для ускорения процессов работы с данными.

Выделяют три основных режима работы приложений, связанных с использованием баз данных.

Режим 1. Получить все данные (последовательная обработка).

Режим 2. Получить уникальные (например, одна запись) данные, для чего используют прямой доступ (хеширование, идентификаторы), индексный метод (первичный ключ), произвольный доступ, последовательный доступ (бинарное В-дерево, В+-дерево).

Режим 3. Получить некоторые (группу записей) данные, для чего применяют вторичные ключи, мультисписок, инвертированный метод, двусвязное дерево.

В БД содержатся не только пользовательские данные, но и большой объем специальных данных. Поэтому в составе БД можно выделить следующие виды данных:

- Пользовательские данные, которые отражают информационные потребности конечных пользователей (именно проектирование моделей этих данных обсуждалось ранее);
- Управляющие данные – метаданные
- Вспомогательные данные.

Для ускорения процесса поиска и упорядочения данных создаются вспомогательные индексные файлы, которые хранят специальные данные – индексы. В качестве индексов могут выступать отдельные поля, прежде всего ключи. Индексный файл меньше по размеру, и потому скорость поиска увеличивается. «Платой» за это является необходимость использования дополнительной, вторичной память. В зависимости от способа построения различают плотный и разреженные индексы. Индекс может быть многоуровневым (В+-деревья). Часто в качестве индексов используют числа.

Основными методами хранения и поиска данных в БД являются:

- последовательный,
- прямой,
- индексно-последовательный
- индексно-прямой.

Для их сравнительной оценки использую различные критерии:

- Эффективность хранения – величина, обратная среднему числу байтов вторичной памяти, необходимому для хранения одного байта исходной памяти.
- Эффективность доступа – величина, обратная среднему числу физических обращений, необходимых для осуществления логического доступа.

При использовании индексно-прямого и индексно-последовательного методов доступа различают основной и индексный файл.

5.3. Способы индексации

Физически последовательный метод. Записи хранятся в логической последовательности, файл имеет постоянный размер, указатели могут отсутствовать. Данные хранятся в главном файле, а обновление требует создания нового главного файла с упорядочением, для чего используется вспомогательный файл. Эффективность использования памяти близка к ста процентам, эффективность доступа низка.

Метод удобен для режима 1, однако быстродействие в режиме 2 мало: для его повышения необходимо использовать бинарный поиск (B- и B+-деревья). Время включения и удаления записей значительно.

Индексно-последовательный метод. Индексный файл упорядочен по первичному ключу (главному атрибуту физической записи). Индекс является разреженным, индексный файл (рис. 5.9) содержит ссылки не на каждую запись, а на группу записей Последовательная организация индексного файла допускает, в свою очередь, его индексацию (многоуровневая индексация). Процедура добавления возможна в двух видах.

1. Новая запись запоминается в отдельном файле (области), называемом областью переполнения. Блок этой области связывается в цепочку с блоком, которому логически принадлежит запись. Запись вводится в основной файл.
2. Если места в блоке основного файла нет, запись делится пополам и в индексном файле создается новый блок.

Наличие индексного файла большого размера снижает эффективность доступа. В большой БД основным параметром становится скорость выборки первичных и вторичных ключей. При большой интенсивности обновления данных следует периодически проводить реорганизацию БД. Эффективность хранения зависит от размера и изменяемости БД, а эффективность доступа - от числа уровней индексации, распределения памяти для хранения индекса, числа записей в БД, уровня переполнения.

При произвольных методах записи физически располагаются произвольно.

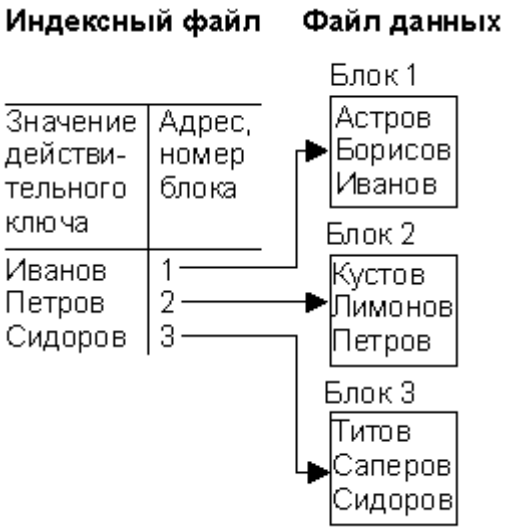

Рис. 5.9. Схема организации индексно-последовательного метода

Индексно-прямой метод. В этом случае индексный файл (рис. 5.10) содержит плотный индекс, при котором каждой записи основного файла запись (индекс) в индексном файле. Записи хранятся в произвольном порядке. Создается отдельный файл, хранящий значение действительного ключа и физического адреса (индекса).

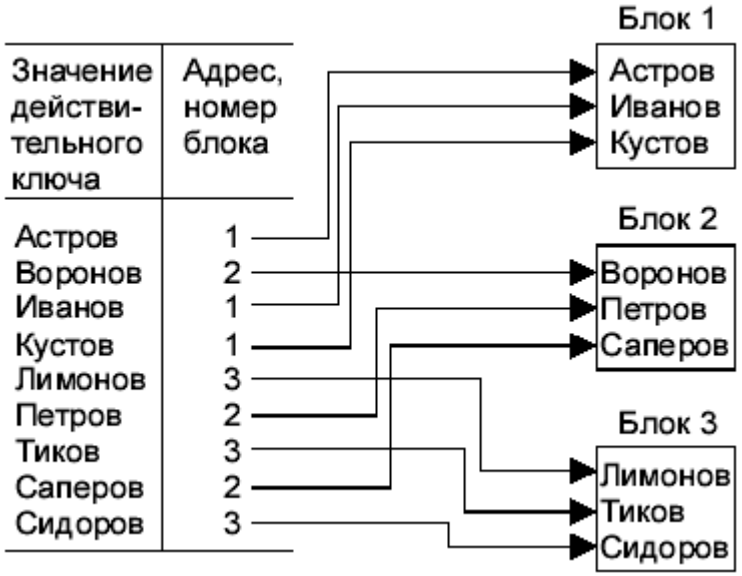

Рис. 5.10. Схема организации индексно-прямого метода доступа

Прямой метод. Имеется взаимно-однозначное соответствие (рис. 5.11) между ключом записи и ее физическим адресом.

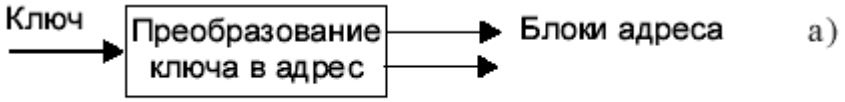

Рис. 5.11. Схема организации прямого метода доступа

В этом методе необходимо преобразование и надо четко знать исходные данные и организацию памяти. Ключи должны быть уникальными.

Эффективность доступа равна 1, а эффективность хранения зависит от плотности ключей. Если не требовать взаимной однозначности, то этот метод называется метод с использованием хеширования - быстрого поиска данных, особенно при добавлении элементов непредсказуемым образом на основе хеш-функции. Хеширование - метод доступа, обеспечивающий прямую адресацию данных путем преобразования значений ключа в относительный или абсолютный физический адрес. Адрес является функцией значения поля и ключа записи. Записи, приобретающие один адрес, называются синонимами. В качестве

критериев оптимизации алгоритма хеширования могут быть: потеря связи между адресом и значением поля, минимум синонимов. Память делится на страницы определенного размера, и все синонимы помещаются в пределах одной страницы или в область переполнения. Механизм поиска синонимов - цепочечный. Любой элемент хеш-таблицы имеет особый ключ, а само занесение осуществляется с помощью хеш-функции, отображающей ключи на множество целых чисел, которые лежат внутри диапазона адресов таблицы.

Хеш-функция должна обеспечивать равномерное распределение ключей по адресам таблицы, однако двум разным ключам может соответствовать один адрес. Если адрес уже занят, возникает состояние, называемое коллизией, которое устраняется специальными алгоритмами: проверка идет к следующей ячейке до обнаружения своей ячейки. Элемент с ключом помещается в эту ячейку.

Для поиска используется аналогичный алгоритм: вычисляется значение хеш-функции, соответствующее ключу, проверяется элемент таблицы, находящийся по указанному адресу. Если обнаруживается пустая ячейка, то элемента нет.

Размер хеш-таблицы должен быть больше числа размещаемых элементов. Если таблица заполняется на шестьдесят процентов, то, как показывает практика, для размещения нового элемента проверяется в среднем не более двух ячеек. После удаления элемента пространство памяти, как правило, не может быть использовано повторно.

Бинарное дерево является разновидностью B-дерева. B-дерево допускает более двух ветвей, исходящих из одной вершины. Любая вершина состоит из совокупности значений первичного ключа, указателей индексов и (ассоциированных) данных. Указатель индекса используется для перехода на следующий, более низкий уровень вершин. «Хранимые» в вершине данные фактически представляют собой совокупность указателей данных и служат для физической организации данных, определения положения данных, ключевое значение которых хранится в этой вершине индекса. Физическая организация ветвящейся вершины B-дерева подобна физической последовательной структуре. Этот метод дает хорошее использование памяти, обладает малым числом подопераций. Включение и удаление данных достаточно просто и эффективно.

Более распространенным вариантом B-дерева является B+-дерево. Здесь возможно использовать двунаправленные указатели, а в промежуточных вершинах имеет место дублирование ключей. Если происходит деление вершины, то в исходную вершину пересылается значение среднего ключа. Фактически B+-дерево есть индекс (указатель всех записей файла) вместе с B-деревом, как многоуровневым указателем на элементы последнего индекса.

Контрольные вопросы

1. Приведите классификацию файлов, используемых в системах БД.
2. Дайте характеристику файлам прямого доступа.
3. Для чего используется область переполнения?
4. Охарактеризуйте стратегию свободного замещения.
5. Назовите элементы физической модели дискового пространства.
6. Определите структуру страницы и дайте характеристику её компонентов.
7. Для чего используется индексный файл?
8. Что такое плотный индекс?
9. Как строится разреженный индекс?
10. Опишите алгоритм поиска данных с использованием плотного индекса.
11. Опишите алгоритм поиска данных с использованием разреженного индекса.
12. Опишите алгоритм добавления данных с использованием плотного индекса.

ЛЕКЦИЯ 6. ОСНОВЫ ЯЗЫКА SQL

Стандарты языков SQL. Интерактивный, встроенный, динамический SQL. Особенности использования SQL в многопользовательской среде: SQL как средство общения в распределенной среде. Операторы SQL. История развития SQL.

В начале 70-х годов после создания реляционной модели Кодда привел к появлению структурированного языка запросов к реляционной БД – SQL. Первые разработки в области теории реляционных БД были проведены компанией IBM, но первой на рынок вышла компания ORACLE. Язык SQL стал обязательным для реляционных СУБД. Американский институт стандартов ANSI разработал стандарты для SQL (1986, 1989, 1992, 1999) , 2003 г.г. В настоящее время разработан стандарт SQL 2008. Он не является свободно доступным. Стандарт SQL 2008 состоит из следующих частей:

- ISO/IEC 9075-1:2008 Framework (SQL/Framework)
- ISO/IEC 9075-2:2008 Foundation (SQL/Foundation)
- ISO/IEC 9075-3:2008 Call-Level Interface (SQL/CLI)
- ISO/IEC 9075-4:2008 Persistent Stored Modules (SQL/PSM)
- ISO/IEC 9075-9:2008 Management of External Data (SQL/MED)
- ISO/IEC 9075-10:2008 Object Language Bindings (SQL/OLB)
- ISO/IEC 9075-11:2008 Information and Definition Schemas (SQL/Schemata)
- ISO/IEC 9075-13:2008 SQL Routines and Types Using the Java TM Programming Language (SQL/JRT)
- ISO/IEC 9075-14:2008 XML-Related Specifications (SQL/XML)

Сейчас существует большое количество реализаций языка, которые имеют специфические отличия, подчиняясь, тем не менее, единому стандарту:

- IBM DB2 Universal DataBase;
- My SQL;
- Oracle DataBase 10g;
- Microsoft SQL Server 2008 и др.

6.1. Способы реализации SQL

SQL является полным языком (в отличии от теоретических языков реляционной алгебры Кодда), включающем ЯОД и ЯМД, а также операторы управления БД.

Существует несколько способов реализации языка: интерактивный, встроенный, динамический. *Интерактивная реализация* предполагает возможность непосредственного задания операторов при работе с СУБД, которые сразу выполняются и выдают результат.

Встроенная реализация предполагает возможность статического использования операторов SQL в программы на ЯВУ: C, COBOL, PL/1, Pascal и др. В тексте программ на этих языках имеются операторы обращения к SQL, которые жестко включаются в выполнимый модуль после компиляции. Фундаментальным принципом технологии встроенного SQL является то, что любое SQL-выражение, которое может быть использовано интерактивно, можно применять и для встроенной реализации. Выполняемый SQL-оператор является полноправным оператором языка, в который встроен SQL, и может встречаться в любом месте программы. Они могут включать ссылки на переменные базового языка. При этом необходимо помнить о соответствии типов переменных программы и типов элементов БД.

При *динамическом использовании* языка предполагается динамическое построение вызовов SQL-функций из прикладных программ и интерпретацией этих функций. Используется тогда, когда в приложении заранее неизвестен вид SQL-вызова, и он строится в диалоге с пользователем.

Постоянное развитие стандарта SQL способствовало появлению среди разных производителей и платформ многочисленных диалектов SQL. Они развиваются благодаря тому, что пользователям конкретной СУБД требуются новые возможности, не предусмотренных стандартом ANSI (например, средства условной обработки IF ... THEN для обработки ошибок). К таким диалектам можно отнести PL/SQL (Oracle), Transact-SQL (Microsoft SQL), PL/pgSQL (PostgreSQL), SQLPL (DB2).

Наиболее популярные серверные СУБД:

- DB2 – СУБД от IBM, работает на различных аппаратных платформах от ПК до больших ЭВМ, под управлением многих ОС (в том числе Linux, UNIX, Windows), распространена в больших корпоративных БД.
- MySQL – популярная СУБД с открытым кодом, работает под управлением многих ОС, в том числе и Linux.
- Oracle – ведущая СУБД в коммерческом секторе, реализована на многих аппаратных платформах и ОС
- SQL Server – работает только под управлением Windows

6.2. Структура SQL

В SQL можно выделить следующие компоненты

- **SQL-DDL** (Data Definition Language) – язык определения структур и ограничений целостности баз данных. Сюда относятся команды создания и удаления баз данных; создания, изменения и удаления таблиц; управления пользователями и т.д.
- **SQL-DML** (Data Manipulation Language) – язык манипулирования данными: добавление, изменение, удаление и извлечение данных, управления транзакциями
    - Операторы программирования
    - Средства администрирования

6.3. Типы данных в SQL

- *Символьные типы данных* – содержат буквы, цифры и специальные символы.
    - **CHAR** или **CHAR(*n*)** – символьные строки фиксированной длины. Длина строки определяется параметром **n.** **CHAR** без параметра соответствует **CHAR(1)**. Для хранения таких данных всегда отводится **n** байт вне зависимости от реальной длины строки.
    - **VARCHAR(n)** – символьная строка переменной длины. Для хранения данных этого типа отводится число байт, соответствующее реальной длине строки.
- *Целые типы данных* – поддерживают только целые числа (дробные части и десятичные точки не допускаются). Над этими типами разрешается выполнять арифметические операции и применять к ним агрегирующие функции (определение максимального, минимального, среднего и суммарного значения столбца реляционной таблицы).
    - **INTEGER** или **INT**– целое, для хранения которого отводится, как правило, 4 байта. (Замечание: число байт, отводимое для хранения того или иного числового типа данных зависит от используемой СУБД и аппаратной платформы, здесь приводятся наиболее "типичные" значения) Интервал значений от - 2147483647 до + 2147483648
    - **SMALLINT** – короткое целое (2 байта), интервал значений от - 32767 до +32768
- *Вещественные типы данных* – описывают числа с дробной частью.
    - **FLOAT** и **SMALLFLOAT** – числа с плавающей точкой (для хранения отводится обычно 8 и 4 байта соответственно).
    - **DECIMAL(p)** – тип данных аналогичный FLOAT с числом значащих цифр p.
    - **DECIMAL(p,n)** - аналогично предыдущему, p - общее количество десятичных цифр, n - количество цифр после десятичной запятой.

- *Денежные типы данных* – описывают, естественно, денежные величины. Его представление аналогично типу данных DECIMAL(p,n).
    - **MONEY(p,n) –** все аналогично типу **DECIMAL(p,n).** Вводится только потому, что некоторые СУБД предусматривают для него специальные методы форматирования.
- *Дата и время* – используются для хранения даты, времени и их комбинаций. Большинство СУБД умеет определять интервал между двумя датами, а также уменьшать или увеличивать дату на определенное количество времени.
    - **DATE** – тип данных для хранения даты.
    - **TIME** – тип данных для хранения времени.
    - **INTERVAL** – тип данных для хранения временного интервала.
    - **DATETIME** – тип данных для хранения моментов времени (год + месяц + день + часы + минуты + секунды + доли секунд).
- *Двоичные типы данных* – позволяют хранить данные любого объема в двоичном коде (оцифрованные изображения, исполняемые файлы и т.д.). Определения этих типов наиболее сильно различаются от системы к системе, часто используются ключевые слова:
    - **BINARY**
    - **BYTE**
    - **BLOB**
- *Последовательные типы данных* – используются для представления возрастающих числовых последовательностей.
    - **SERIAL** – тип данных на основе **INTEGER**, позволяющий сформировать уникальное значение (например, для первичного ключа). При добавлении записи СУБД автоматически присваивает полю данного типа значение, получаемое из возрастающей последовательности целых чисел.

6.4. Опреаторы ЯОД SQL

6.4.1. Оператор CREATE TABLE

Оператор используется для создания новых таблиц базы данных.

CREATE TABLE имя_таблицы (
имя_столбца тип_данных [NULL | NOT NULL] [CONSTRAINTS],
имя_столбца тип_данных[NULL|NOT NULL] [CONSTRAINTS] , ...);

```
CREATE TABLE
   [ database_name . [ schema_name ] . | schema_name . ] table_name
      ( { <column_definition> | <computed_column_definition> }
      [ <table_constraint> ] [ ,...n ] )
   [ ON { partition_scheme_name ( partition_column_name ) | filegroup
      | "default" } ]
   [ { TEXTIMAGE_ON { filegroup | "default" } ]
[ ; ]

<column_definition> ::=
column_name <data_type>
   [ COLLATE collation_name ]
   [ NULL | NOT NULL ]
   [
      [ CONSTRAINT constraint_name ] DEFAULT constant_expression ]
    | [ IDENTITY [ ( seed , increment ) ] [ NOT FOR REPLICATION ]
   ]
   [ ROWGUIDCOL ] [ <column_constraint> [ ...n ] ]

<data type> ::=
[ type_schema_name . ] type_name
   [ ( precision [ , scale ] | max |
      [ { CONTENT | DOCUMENT } ] xml_schema_collection ) ]
<column_constraint> ::=
[ CONSTRAINT constraint_name ]
```

```
{    { PRIMARY KEY | UNIQUE }
       [ CLUSTERED | NONCLUSTERED ]
       [
          WITH FILLFACTOR = fillfactor
        | WITH ( < index_option > [ , ...n ] )
       ]
       [ ON { partition_scheme_name ( partition_column_name )
          | filegroup | "default" } ]
  | [ FOREIGN KEY ]
       REFERENCES [ schema_name . ] referenced_table_name [ ( ref_column ) ]
       [ ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
       [ ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
       [ NOT FOR REPLICATION ]
  | CHECK [ NOT FOR REPLICATION ] ( logical_expression )
}

<computed_column_definition> ::=
column_name AS computed_column_expression
[ PERSISTED [ NOT NULL ] ]
[
   [ CONSTRAINT constraint_name ]
   { PRIMARY KEY | UNIQUE }
      [ CLUSTERED | NONCLUSTERED ]
      [
         WITH FILLFACTOR = fillfactor
       | WITH ( <index_option> [ , ...n ] )
      ]
      [ ON { partition_scheme_name ( partition_column_name )
         | filegroup | "default" } ]
   | [ FOREIGN KEY ]
      REFERENCES referenced_table_name [ ( ref_column ) ]
      [ ON DELETE { NO ACTION | CASCADE } ]
```

```
      [ ON UPDATE { NO ACTION } ]
      [ NOT FOR REPLICATION ]
   | CHECK [ NOT FOR REPLICATION ] ( logical_expression )
]

< table_constraint > ::=
[ CONSTRAINT constraint_name ]
{
   { PRIMARY KEY | UNIQUE }
      [ CLUSTERED | NONCLUSTERED ]
            (column [ ASC | DESC ] [ ,...n ] )
      [
         WITH FILLFACTOR = fillfactor
        |WITH ( <index_option> [ , ...n ] )
      ]
      [ ON { partition_scheme_name (partition_column_name)
         | filegroup | "default" } ]
   | FOREIGN KEY
            ( column [ ,...n ] )
      REFERENCES referenced_table_name [ ( ref_column [ ,...n ] ) ]
      [ ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
      [ ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
      [ NOT FOR REPLICATION ]
   | CHECK [ NOT FOR REPLICATION ] ( logical_expression )
}
<index_option> ::=
{
   PAD_INDEX = { ON | OFF }
  | FILLFACTOR = fillfactor
  | IGNORE_DUP_KEY = { ON | OFF }
  | STATISTICS_NORECOMPUTE = { ON | OFF }
  | ALLOW_ROW_LOCKS = { ON | OFF}
```

```
  | ALLOW_PAGE_LOCKS ={ ON | OFF}
}
```

6.4.2. Оператор ALTER TABLE

Оператор используется для обновления схемы существующей таблицы.

```
ALTER TABLE имя таблицы

ADD | DROP имя_столбца тип_данных [NULL | NOT 1>NULL] [CONSTRAINTS] ,
ADD I DROP имя_столбца тип_данных [NULL|NOT 4>NULL] [CONSTRAINTS] ,
```

6.4.3. Оператор CREATE INDEX

Оператор используется для создания индекса одного или нескольких столбцов.

Синтаксис

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX index_name
   ON <object> ( column [ ASC | DESC ] [ ,...n ] )
   [ INCLUDE ( column_name [ ,...n ] ) ]
   [ WITH ( <relational_index_option> [ ,...n ] ) ]
   [ ON { partition_scheme_name ( column_name )
        | filegroup_name
        | default
        }
   ]
[ ; ]

<object> ::=
{
   [ database_name. [ schema_name ] . | schema_name. ]
      table_or_view_name
}

<relational_index_option> ::=
{
```

```
   PAD_INDEX  = { ON | OFF }
  | FILLFACTOR = fillfactor
  | SORT_IN_TEMPDB = { ON | OFF }
  | IGNORE_DUP_KEY = { ON | OFF }
  | STATISTICS_NORECOMPUTE = { ON | OFF }
  | DROP_EXISTING = { ON | OFF }
  | ONLINE = { ON | OFF }
  | ALLOW_ROW_LOCKS = { ON | OFF }
  | ALLOW_PAGE_LOCKS = { ON | OFF }
  | MAXDOP = max_degree_of_parallelism
}

Create XML Index
CREATE [ PRIMARY ] XML INDEX index_name
   ON <object> ( xml_column_name )
   [ USING XML INDEX xml_index_name
      [ FOR { VALUE | PATH | PROPERTY } ] ]
   [ WITH ( <xml_index_option> [ ,...n ] ) ]
[ ; ]

<object> ::=
{
   [ database_name. [ schema_name ] . | schema_name. ]
      table_name
}

<xml_index_option> ::=
{
   PAD_INDEX  = { ON | OFF }
  | FILLFACTOR = fillfactor
  | SORT_IN_TEMPDB = { ON | OFF }
  | STATISTICS_NORECOMPUTE = { ON | OFF }
```

```
  | DROP_EXISTING = { ON | OFF }
  | ALLOW_ROW_LOCKS = { ON | OFF }
  | ALLOW_PAGE_LOCKS = { ON | OFF }
  | MAXDOP = max_degree_of_parallelism
}


CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX index_name
    ON <object> ( column_name [ ASC | DESC ] [ ,...n ] )
    [ WITH <backward_compatible_index_option> [ ,...n ] ]
    [ ON { filegroup_name | "default" } ]

<object> ::=
{
    [ database_name. [ owner_name ] . | owner_name. ]
        table_or_view_name
}

<backward_compatible_index_option> ::=
{
    PAD_INDEX
  | FILLFACTOR = fillfactor
  | SORT_IN_TEMPDB
  | IGNORE_DUP_KEY
  | STATISTICS_NORECOMPUTE
  | DROP_EXISTING
}
```

Аргументы

UNIQUE

Создает уникальный индекс для таблицы или представления. Уникальным является индекс, в котором не может быть двух строк с одним и тем же значением ключа индекса. Кластеризованный индекс представления должен быть уникальным.

Компонент SQL Server 2005 Database Engine не позволяет создать уникальный индекс по столбцам, уже содержащим повторяющиеся значения, даже если параметру IGNORE_DUP_KEY присвоено значение ON. При попытке создания такого индекса компонент Database Engine выдает сообщение об ошибке. Прежде чем создавать уникальный индекс по такому столбцу или столбцам, необходимо удалить все повторяющиеся значения. Столбцы, используемые в уникальном индексе, должны иметь свойство NOT NULL, т. к. при создании индекса значения NULL рассматриваются как повторяющиеся.

CLUSTERED

Создает индекс, в котором логический порядок значений ключа определяет физический порядок соответствующих строк в таблице. На нижнем (конечном) уровне такого индекса хранятся действительные строки данных таблицы. Для таблицы или представления в каждый момент времени может существовать только один кластеризованный индекс. Представление с уникальным кластеризованным индексом называется индексированным. Создание уникального кластеризованного индекса физически материализует представление. Уникальный кластеризованный индекс для представлении должен быть создан до того, как для этого же представления будут определены какие-либо другие индексы. Следует создавать кластеризованные индексы до создания любых некластеризованных. При создании кластеризованного индекса все существующие некластеризованные индексы таблицы перестраиваются.

Если аргумент CLUSTERED не указан, создается некластеризованный индекс.

NONCLUSTERED

Создается индекс, который задает логический порядок таблицы. Логический порядок строк данных в некластеризованном индексе не влияет на их физический порядок. Для каждой таблицы можно создать до 249 некластеризованных индексов, независимо от того, каким образом они создаются: неявно с помощью ограничений PRIMARY KEY и UNIQUE или явно с помощью инструкции CREATE INDEX.

Для индексированных представлений некластеризованные индексы могут создаваться только в случае, если уже определен уникальный кластеризованный индекс. По умолчанию, используется значение NONCLUSTERED.

index_name

Имя индекса. Имена индексов должны быть уникальными в пределах таблицы или представления, но не базы данных. Имена индексов должны соответствовать правилам для иедентификаторов. Имена первичных XML-индексов не должны начинаться со следующих символов: #, ##, @ или @@.

column

Столбец или столбцы, на которых основан индекс. Задайте несколько имен столбцов для создания составного индекса по объединенным значениям указанных столбцов. Столбцы, которые должны быть включены в составной индекс, указываются в скобках за аргументом *table_or_view_name* в порядке сортировки.

В один составной ключ индекса могут входить до 16 столбцов. Все столбцы составного ключа индекса должны находиться в одной таблице или одном и том же представлении. Максимальный общий размер значений составного индекса равен 900 байт. Дополнительные сведения о столбцах переменной длины в составных индексах см. в разделе «Примечания».

Столбцы с типами данных для больших объектов **ntext**, **text**, **varchar(max)**, **nvarchar(max)**, **varbinary(max)**, **xml** или **image** не могут быть ключевыми столбцами для индекса. Кроме того, определение представления не может включать столбцы типов **ntext**, **text** и **image**, даже если они указаны в инструкции CREATE INDEX.

Можно создавать индексы на столбцах с пользовательским типом данных CLR, если этот тип поддерживает двоичное упорядочение. Можно также создавать индексы на вычисляемых столбцах, определенных как вызовы методов для столбцов с пользовательскими типами данных, если эти методы помечены как детерминированные и не выполняют операции доступа к данным.

[ASC | DESC]

Определяет сортировку значений заданного столбца индекса: по возрастанию или по убыванию. По умолчанию, используется аргумент ASC.

INCLUDE **(** *column* [**,**... *n*] **)**

Указывает неключевые столбцы, добавляемые на конечный уровень некластеризованного индекса. Некластеризованный индекс может быть уникальным или неуникальным.

Максимальное количество включенных неключевых столбцов – 1023, минимальное – 1.

В списке INCLUDE имена столбцов не могут повторяться и не могут использоваться одновременно как ключевые и неключевые.Допускаются данные всех типов, кроме **text**, **ntext** и **image**. Индекс должен создаваться или перестраиваться в автономном режиме (ONLINE = OFF), если любой из заданных неключевых столбцов имеет тип данных **varchar(max)**, **nvarchar(max)** или **varbinary(max)**. Включенными столбцами могут быть детерминированные вычисляемые столбцы — как точные, так и неточные. Вычисляемые столбцы, производные от типов данных **image**, **ntext**, **text**, **varchar(max)**, **nvarchar(max)**, **varbinary(max)** и **xml**, могут включаться как неключевые, если типы данных вычисляемых столбцов приемлем для включения. ON *partition_scheme_name* **(** *column_name* **)**

Задает схему секционирования, которая определяет файловые группы соответствующие секциям секционированного индекса. Схема секционирования должна быть уже создана в базе данных с помощью инструкции CREATE PARTITION SCHEME или ALTER PARTITION SCHEME. Аргумент *column_name* задает столбец, по которому будет секционирован индекс. Этот столбец должен соответствовать типу данных, длине и точности аргумента функции секционирования, которую использует схема *partition_scheme_name*. Аргумент *column_name* может указывать на столбцы, не входящие в определение индекса. Можно указать любой столбец базовой таблицы, за исключением случая секционирования индекса UNIQUE, когда столбец *column_name* должен быть выбран из используемых в уникальном ключе. Это ограничение дает возможность компоненту Database Engine проверять уникальность значений ключа только в одной секции.

Если аргумент *partition_scheme_name* или *filegroup* не задан и таблица секционирована, индекс помещается в ту же схему секционирования и с тем же столбцом секционирования, что и для базовой таблицы.

Для XML-индекса задать схему секционирования невозможно. Если базовая таблица секционирована, XML-индекс использует ту же схему секционирования, что и таблица.

ON *filegroup_name*

Создает заданный индекс в указанной файловой группе. Если местоположение не указано и таблица или представление не секционированы, индекс использует ту же файловую группу, что и базовые таблица или представление. Файловая группа уже должна существовать. XML-индексы используют ту же файловую группу, что и таблица.

ON "default"

Создает заданный индекс в файловой группе, используемой по умолчанию.

Слово «default» в этом контексте не является ключевым. Это идентификатор файловой группы, используемой по умолчанию, который должен быть ограничен, как например в выражении ON "default" или ON **[**default**]**. Если задано значение «default», параметр QUOTED_IDENTIFIER в текущем сеансе должен иметь значение ON. Это установка по умолчанию.

[PRIMARY] XML

Создает XML-индекс по заданному столбцу типа **xml**. Если присутствует ключевое слово PRIMARY, создается кластеризованный индекс с ключом, образованным из ключа кластеризации таблицы пользователя и идентификатора XML-узла. Для каждой таблицы можно создать до 249 XML-индексов. При создании XML-индекса помните следующее.

- Кластеризованный индекс должен существовать для первичного ключа таблицы пользователя.
- Максимальное количество столбцов в ключе кластеризации таблицы пользователя — 15.

- У каждого столбца типа **xml** в таблице может быть один первичный XML-индекс и несколько вторичных.
- Чтобы создать вторичный XML-индекс для столбца типа **xml**, первичный XML-индекс для этого столбца уже должен существовать.
- XML-индекс может быть создан только для одного столбца типа **xml**. Невозможно создать XML-индекс для столбца, не относящегося к типу **xml**, а также реляционный индекс для столбца типа **xml.**
- Невозможно создать первичный или вторичный XML-индекс для столбца типа **xml** в представлении для переменной со столбцами типа **xml**, возвращающей табличное значение, или для переменных типа **xml.**
- Невозможно создать первичный XML-индекс для вычисляемого столбца типа **xml.**
- Значения параметров SET должны быть теми же, что и для индексированных представлений и индексов вычисляемых столбцов. В частности, параметр ARITHABORT должен быть в значении ON, если создается XML-индекс и если выполняется вставка, удаление или обновление значений в столбце типа **xml.**

Инструкция CREATE INDEX оптимизируется, как и любой другой запрос. Для уменьшения числа операций ввода-вывода обработчик запросов может вместо таблицы просматривать другой индекс. В некоторых ситуациях можно отказаться от операций сортировки. На многопроцессорных компьютерах с установленным сервером SQL Server 2005 Enterprise Edition инструкция CREATE INDEX, как и другие запросы, может использовать несколько процессоров для операций просмотра и сортировки, связанных с созданием индекса.

6.4.4. Оператор DROP

Оператор DROP навсегда удаляет объекты базы данных (таблицы, представления, индексы и т.д.)

DROP INDEX имя индекса| PROCEDURE имя процедуры|TABLE имя таблицы|VIEW имя представления .

6.5. Опреаторы ЯМД SQL

6.5.1. Оператор INSERT

Оператор добавляет в таблицу одну строку

INSERT INTO имя_таблицы [(имена_столбцов, ...)] VALUES[(значения, ...);

```
INSERT
   [ TOP ( expression ) [ PERCENT ] ]
   [ INTO]
   { <object> | rowset_function_limited
     [ WITH ( <Table_Hint_Limited> [ ...n ] ) ]
   }
{
   [ ( column_list ) ]
   [ <OUTPUT Clause> ]
   { VALUES ( { DEFAULT | NULL | expression } [ ,...n ] )
   | derived_table
   | execute_statement
   }
}
   | DEFAULT VALUES
[; ]

<object> ::=
{
   [ server_name . database_name . schema_name .
     | database_name .[ schema_name ] .
     | schema_name .
   ]
      table_or_view_name
}
```

Аргументы

WITH <common_table_expression>

Определяет временный именованный результирующий набор, также называемый обобщенным табличным выражением, определенным в области инструкции INSERT. Результирующий набор получается из инструкции SELECT.

Обобщенные табличные выражения также используются инструкциями SELECT, DELETE, UPDATE и CREATE VIEW.

TOP **(** *expression* **)** [PERCENT]

Задает количество или процент случайных строк для вставки. Выражение *expression* может быть либо количеством, либо процентом строк. Строки, на которые ссылается выражение TOP, используемое с INSERT, UPDATE и DELETE, не упорядочены.

В инструкциях INSERT, UPDATE и DELETE необходимо разделять круглыми скобками аргумент *expression* в выражении TOP.

INTO

Необязательное ключевое слово, которое можно использовать между ключевым словом INSERT и целевой таблицей.

server_name

Имя сервера (используется функцией OPENDATASOURCE как имя сервера), на котором находится таблица или представление. Если указан аргумент *server_name*, также необходимо указать аргументы *database_name* и *schema_name*.

database_name

Имя базы данных.

schema_name

Имя схемы, к которой принадлежит таблица или представление.

table_or view_name

Имя таблицы или представления, которые принимают данные.

Переменную **table** внутри своей области можно использовать как имя исходной таблицы в инструкции INSERT.

Представление, на которое ссылается аргумент *table_or_view_name*, должно быть обновляемым и ссылаться ровно на одну базовую таблицу в предложении FROM данного представления. Например, инструкция INSERT в многотабличном представлении должна использовать аргумент *column_list*, который ссылается только на столбцы из одной базовой таблицы..

rowset_function_limited

Либо функция OPENQUERY либо функция OPENROWSET.

WITH **(** <table_hint_limited> [... *n*] **)**

Указывает одну или несколько табличных подсказок, разрешенных для целевой таблицы. Необходимо использовать ключевое слово WITH и круглые скобки.

(*column_list* **)**

Список из одного или нескольких столбцов, в которые нужно вставить данные. Аргумент *column_list* должен быть заключен в круглые скобки и разделен запятыми.

Если столбец не внесен в *column_list*, то компонент SQL Server 2005 Database Engine должен обеспечить значение, основанное на определении столбца; в противном случае строку нельзя будет загрузить. Компонент Database Engine автоматически задает значение для столбца, если столбец имеет следующие характеристики.

- Имеется свойство IDENTITY. Используется следующее значение приращения для идентификатора.
- Имеется стандартное значение. Используется стандартное значение для столбца.
- Имеет тип данных **timestamp**. В этом случае используется текущее значение timestamp.
- Неопределенное значение. Используется значение Null.
- Вычисляемый столбец. Используется вычисленное значение.

Аргумент *column_list* и список VALUES необходимо использовать, когда в столбец идентификаторов вставляются явно заданные значения, а параметру SET IDENTITY_INSERT необходимо присвоить значение ON для таблицы.

Предложение OUTPUT возвращает вставленные строки во время операции вставки. Оно не поддерживается инструкциями DML, которые ссылаются на локальные секционированные представления, распределенные секционированные представления, расположенные удаленно таблицы или инструкции INSERT, содержащие аргумент *execute_statement*.

VALUES

Ввод списка со значениями данных для вставки. Для каждого столбца в *column_list*, если этот параметр указан или присутствует в таблице, должно быть одно значение. Список значений должен быть заключен в круглые скобки.

Если значения в списке VALUES идут в порядке, отличном от порядка следования столбцов в таблице, или не для каждого столбца таблицы определено значение, то необходимо использовать аргумент *column_list* для явного указания столбца для хранения каждого входного значения.

DEFAULT

Указывает компоненту Database Engine необходимость принудительно загружать значения по умолчанию, определенные для столбца. Если для столбца не задано значение по умолчанию и он может содержать значение NULL, вставляется значение NULL. В столбцы с типом данных **timestamp** вставляется следующее значение временной метки. Значение DEFAULT недопустимо для столбца идентификаторов.

expression

Константа, переменная или выражение. В выражении не может содержаться инструкция SELECT или EXECUTE.

derived_table

Любая допустимая инструкция SELECT, возвращающая строки данных, которые загружаются в таблицу. Инструкция SELECT не может содержать обобщенное табличное выражение (CTE).

execute_statement

Любая допустимая инструкция EXECUTE, возвращающая данные с помощью инструкций SELECT или READTEXT. Инструкция SELECT не может содержать CTE-выражение.

Если аргумент *execute_statement* используется с инструкцией INSERT, каждый результирующий набор должен быть совместим со столбцами в таблице или списке *column_list*.

Аргумент *execute_statement* может применяться для выполнения хранимых процедур на том же сервере или на сервере, расположенном удаленно. На удаленном сервере выполняется процедура, результирующий набор возвращается на локальный сервер и загружается в таблицу на локальном сервере.

В SQL Server 2008 изменена семантика транзакций инструкций INSERT...EXECUTE, выполняемых на связанном сервере с замыканием на себя. В SQL Server 2005 этот сценарий не поддерживается и приводит к ошибкам. В SQL Server 2008 инструкция INSERT...EXECUTE может применяться к связанному серверу с замыканием на себя, когда для

соединения не включен режим MARS. Если режим MARS не включен для соединения, то поведение такое же, как в SQL Server 2005.

Если аргумент *execute_statement* возвращает данные с инструкцией READTEXT, необходимо учитывать, что каждая инструкция READTEXT может возвращать не более 1 МБ (1024 КБ) данных. Аргумент *execute_statement* также может использоваться с расширенными процедурами. В этом случае он вставляет данные, возвращенные основным потоком расширенной процедуры, но выходные данные, возвращенные потоками, отличными от основного, не будут вставлены.

DEFAULT VALUES

Заполняет новую строку значениями по умолчанию, определенными для каждого столбца.

Замечания

Инструкция INSERT добавляет новые строки к таблице. Чтобы заменить данные в таблице, перед загрузкой новых данных с помощью инструкции INSERT необходимо применить инструкции DELETEили TRANCATE TABLE для очистки существующих данных.

Столбцы, созданные с типом данных **uniqueidentifier**, содержат двоичные 16-байтные величины специального формата. В отличие от столбцов идентификаторов компонента Database Engine не создает автоматически значения для столбцов с типом данных **uniqueidentifier**. Во время операции вставки переменные с типом данных **uniqueidentifier** и строковые константы вида *xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx* (36 символов, включая дефисы, где *x* – шестнадцатеричная цифра в диапазоне от 0-9 или a-f) можно использовать для столбцов **uniqueidentifier**. Например, 6F9619FF-8B86-D011-B42D-00C04FC964FF является допустимым значением переменной **uniqueidentifier** или столбца. Используйте функцию NEWID() для получения глобального уникального идентификатора (идентификатор GUID).

Инструкции INSERT не учитывает настройки параметра SET ROWCOUNT в местных и расположенных удаленно секционированных представлениях. Также этот параметр не поддерживается инструкциями INSERT для удаленных таблиц в компоненте Database Engine в случае, если уровень совместимости 80 или выше.

Если при выполнении инструкции INSERT возникает арифметическая ошибка (переполнение, деление на ноль или ошибка домена), компонент Database Engine обрабатывает эти ошибки так же, как если бы параметру SET ARITHABORT было присвоено значение ON. Выполнение пакета прекращается и выводится сообщение об ошибке.

При вставке строк применяются следующие правила.

- Если значение загружается в столбцы с типом данных **char**, **varchar** или **varbinary**, то дополнение или усечение конечных пробелов (пробелы для **char** и **varchar**, нули для **varbinary**) определяет параметр SET

ANSI_PADDING, определенный для столбца при создании таблицы. В таблице 6.1. показаны операции по умолчанию для параметра SET ANSI_PADDING, установленного в значение OFF.

Таблица 6.1.

Тип данных	**Стандартная операция**
char	Заполнение значения пробелами до заданной ширины столбца.
varchar	Удаление конечных пробелов до последнего ненулевого символа или до одного пробела, если строка состоит только из пробелов.
varbinary	Удаление конечных нулей.

- Если пустая строка ('') загружена в столбец с типом данных **varchar** или **text**, то операцией по умолчанию будет загрузка строки нулевой длины.
- Если инструкция INSERT нарушает ограничение или правило или если в ней содержится значение, несовместимое с типом данных столбца, то инструкция не выполняется и компонент Database Engine выдает сообщение об ошибке.
- Вставка значения NULL в столбец **text** или **image** не приводит ни к созданию допустимого текстового указателя, ни к предварительному распределению 8-килобайтной текстовой страницы.
- Если инструкция INSERT загружает несколько строк с помощью инструкций SELECT или EXECUTE, то любые нарушения правил или ограничений, возникающие из-за загружаемых значений, приводят к остановке выполнения всей инструкции, и ни одна из строк не будет загружена.
- Если во время вставки значений в таблицы удаленного экземпляра компонента Database Engine указаны не все значения для всех столбцов, необходимо указать столбцы, в которые вставляются определенные значения.

6.5.2. Оператор UPDATE

Оператор обновляет одну или несколько строк таблицы.

UPDATE имя_таблицы SET имя_столбца = значение, [WHERE ...];

UPDATE (column)

Аргументы

column

Это имя столбца для проверки на действие INSERT или UPDATE. Так как имя столбца указано в предложении триггера ON, не ставьте имя таблицы перед именем столбца. Столбец может содержать любой тип данных, поддерживаемый SQL Server 2005. Однако вычисляемые столбцы не могут использоваться в данном контексте.

Типы возвращаемых данных

Boolean

Замечания

Функция UPDATE() возвращает TRUE независимо от того, была ли попытка применить операторы INSERT или UPDATE удачной.

Чтобы проверить действие операторов INSERT или UPDATE для нескольких столбцов, укажите отдельно предложение UPDATE(*column*), следующее за первым предложением. Несколько столбцов также могут быть проверены на действие INSERT или UPDATE при помощи COLUMNS_UPDATED. В результате возвращается битовый шаблон, который указывает на то, какие столбцы были вставлены или обновлены.

IF UPDATE возвращает значение TRUE по действиям оператора INSERT, так как столбцы содержат либо явные вставленные значения, либо неявные вставленные значения (NULL).

UPDATE(*column*) может применяться в любой части тела триггера Transact-SQL.

Примеры

6.5.3. Оператор DELETE

Оператор удаляет одну или несколько строк таблицы

```
DELETE FROM имя_таблицы [WHERE ...];
[ WITH <common_table_expression> [ ,...n ] ]
DELETE
   [ TOP ( expression ) [ PERCENT ] ]
   [ FROM ]
      { table_name [ WITH ( <table_hint_limited> [ ...n ] ) ]
      | view_name
      | rowset_function_limited
      | table_valued_function
   }
   [ <OUTPUT Clause> ]
   [ FROM <table_source> [ ,...n ] ]
   [ WHERE { <search_condition>
         | { [ CURRENT OF
               { { [ GLOBAL ] cursor_name }
                  | cursor_variable_name
               }
            ]
          }
         }
   ]
   [ OPTION ( <Query Hint> [ ,...n ] ) ]
[; ]

<object> ::=
{
   [ server_name.database_name.schema_name.
     | database_name. [ schema_name ] .
     | schema_name.
```

]
table_or_view_name
}
Аргументы
WITH <common_table_expression>

Задает временный именованный результирующий набор, также называемый обобщенным табличным выражением, который определяется в области действия инструкции DELETE. Результирующий набор получается из инструкции SELECT.

Обобщенные табличные выражения также можно использовать в инструкциях SELECT, INSERT, UPDATE и CREATE VIEW..

TOP **(** *expression* **)** [PERCENT]

Задает число или процент случайных строк для удаления. Выражение *expression* может быть либо числом, либо процентом строк. Строки, на которые ссылается выражение TOP, используемое с инструкциями INSERT, UPDATE и DELETE, не упорядочиваются.

Разделение круглыми скобками выражения *expression* в выражении TOP требуется в инструкциях INSERT, UPDATE и DELETE.

FROM

Необязательное ключевое слово, которое можно использовать между ключевым словом DELETE и целевым аргументом *table_or_view_name* или *rowset_function_limited*.

server_name

Имя сервера (с использованием имени связанного сервера или функции OPENDATASOUTCE в качестве имени сервера), на котором расположена таблица или представление. Если аргумент *server_name* указывается, аргументы *database_name* и *schema_name* обязательны.

database_name

Имя базы данных.

schema_name

Имя схемы, которой принадлежит таблица или представление.

table

Имя таблицы, из которой удаляются строки.

Переменную **table** в пределах ее области действия также можно использовать в качестве источника таблицы в инструкции DELETE.

WITH **(** <table_hint_limited> [... *n*] **)**

Задает одну или несколько табличных подсказок, разрешенных для целевой таблицы. Ключевое слово WITH и круглые скобки обязательны. Использование ключевых слов NOLOCK и READUNCOMMITTED запрещено..

view_name

Имя представления, из которого удаляются строки.

Представление, на которое ссылается аргумент *view_name* , должно быть обновляемым и ссылаться ровно на одну базовую таблицу в предложении FROM данного представления..

rowset_function_limited

Функция OPENQUERY или OPENROWSET в зависимости от возможностей поставщика.

table_valued_function

Может быть возвращающей табличное значение функцией.

<OUTPUT_Clause>

Возвращает удаленные строки или выражения, основанные на них, как часть операции DELETE. Предложение OUTPUT не поддерживается ни в каких инструкциях DML, направленных на представления и удаленные таблицы..

FROM <table_source>

Задает дополнительное предложение FROM. Это расширение языка Transact-SQL для инструкции DELETE позволяет задавать данные из <table_source> и удалять соответствующие строки из таблицы в первом предложении FROM.

Это расширение, в котором задается соединение, может быть использовано вместо вложенного запроса в предложении WHERE для указания удаляемых строк.

WHERE

Указывает условия, используемые для ограничения числа удаляемых строк. Если предложение WHERE не указывается, инструкция DELETE удаляет все строки из таблицы.

Предусмотрено два вида операций удаления в соответствии с тем, что указывается в предложении WHERE.

- Операции удаления с поиском указывают условие поиска для уточнения строк, которые будут удалены. Например, WHERE *column_name = value*.
- Операции удаления по позиции используют предложение CURRENT OF для указания курсора. Удаление осуществляется в текущей позиции курсора. Эта операция может быть более точной, чем инструкция DELETE по найденному, которая использует предложение WHERE *search_condition* для указания удаляемых строк. Инструкция DELETE по найденному удаляет несколько строк, если условие поиска не определяет уникально одну строку.

<search_condition>

Указывает ограничивающие условия для удаляемых строк. Количество предикатов, которое может содержать условие поиска, не ограничено.

CURRENT OF

Указывает выполнение инструкции DELETE в текущей позиции указанного курсора.

GLOBAL

Указывает, что аргумент *cursor_name* ссылается на глобальный курсор.

cursor_name

Имя открытого курсора, из которого производится выборка. Если существует как глобальный, так и локальный курсор с именем *cursor_name*, то, когда указывается GLOBAL, этот аргумент указывает на глобальный курсор; в противном случае — на локальный курсор. Курсор должен позволять производить обновления.

cursor_variable_name

Имя переменной курсора. Переменная курсора должна содержать ссылку на курсор, обновления которого разрешены.

OPTION **(** <query_hint> [**,**... *n*] **)**

Ключевые слова, показывающие, что подсказки оптимизатора применяются при настройке способа обработки инструкции компонентом Database Engine..

Замечания

Инструкцию DELETE можно использовать в теле пользовательской функции, если изменяемым объектом является переменная **table.**

При выполнении инструкции DELETE может произойти ошибка, если она нарушает триггер или пытается удалить строку, на которую ссылаются данные в другой таблице с помощью ограничения FOREIGN KEY. Если инструкция DELETE удаляет несколько строк и одна из удаленных строк нарушает триггер или ограничение, то эта инструкция отменяется, т.е. возвращается ошибка и строки не удаляются.

В случае арифметической ошибки (переполнение, деление на ноль или выход за пределы допустимых значений), возникающей в ходе вычисления выражения при выполнении инструкции DELETE, компонент Database Engine будет обрабатывать эти ошибки, как если бы параметр SET ARITHABORT имел значение ON. Оставшаяся часть пакетной операции отменяется и возвращается сообщение об ошибке.

Значение параметра SET ROWCOUNT не учитывается в инструкциях DELETE для удаленных таблиц и локальных и удаленных секционированных представлений.

6.5.4. Оператор SELECT

Извлекает строки из базы данных и позволяет делать выборку одной или нескольких строк или столбцов из одной или нескольких таблиц. Полный синтаксис инструкции SELECT сложен, однако основные предложения можно вкратце описать следующим образом:

[WITH <common_table_expression>]

SELECT *select_list* [INTO *new_table*]

[FROM *table_source*] [WHERE *search_condition*]

[GROUP BY *group_by_expression*]

[HAVING *search_condition*]

[ORDER BY *order_expression* [ASC | DESC]]

Полный синтаксис

```
SELECT statement ::=
   [WITH <common_table_expression> [,...n]]
   <query_expression>
   [ ORDER BY { order_by_expression | column_position [ ASC | DESC ] }
  [ ,...n ] ]
   [ COMPUTE
  { { AVG | COUNT | MAX | MIN | SUM } ( expression ) } [ ,...n ]
  [ BY expression [ ,...n ] ]
   ]
   [ <FOR Clause>]
   [ OPTION ( <query_hint> [ ,...n ] ) ]
<query_expression> ::=
   { <query_specification> | ( <query_expression> ) }
   [  { UNION [ ALL ] | EXCEPT | INTERSECT }
      <query_specification> | ( <query_expression> ) [...n ] ]
<query_specification> ::=
SELECT [ ALL | DISTINCT ]
   [TOP expression [PERCENT] [ WITH TIES ] ]
   < select_list >
   [ INTO new_table ]
   [ FROM { <table_source> } [ ,...n ] ]
   [ WHERE <search_condition> ]
```

[GROUP BY [ALL] group_by_expression [,...n]
[WITH { CUBE | ROLLUP }]
]
[HAVING < search_condition >]

Контрольные вопросы

1. Для чего используется стандартизация языков работы с СУБД?
2. В чем отличие интерактивной и встроенной реализации языка SQL?
3. Определите структуру языка SQL.
4. Какие типы данных определены в стандарте языка SQL?
5. Назовите операторы ЯОД.
6. Для чего используется описатель Null в операторе CREATE TABLE?
7. Определите синтаксис оператора ALTER TABLE.
8. Какие виды индекса можно определить при использовании оператора CREATE INDEX?
9. Назовите операторы ЯМД.
10. Для чего используется оператор INSERT?
11. Определите синтаксис оператора UPDATE.
12. Для чего используется оператор DELETE?
13. Назовите основные группы опций оператора SELECT.
14. Для чего используется опция GROUP BY?
15. Определите синтаксические правила задания условия отбора в опции WHERE оператора SELECT.

ЛЕКЦИЯ 7. ХРАНИМЫЕ ПРОЦЕДУРЫ И ТРИГГЕРЫ

Назначение и использование хранимых процедур. Средства поддержания динамической целостности. Хранимые процедуры и триггеры.

7.1. Хранимая процедура

7.1.1. Назначение хранимых процедур

Хранимая процедура – это набор операторов T-SQL, который компилируется системой SQL Server в единый "план исполнения". Этот план сохраняется в кэш-области памяти для процедур при первом выполнении хранимой процедуры, что позволяет использовать этот план повторно; системе SQL Server не требуется снова компилировать эту процедуру при каждом ее запуске. Хранимые процедуры T-SQL аналогичны процедурам в других языках программирования в том смысле, что они допускают входные параметры и возвращают выходные значения в виде параметров или сообщения о состоянии (успешное или неуспешное завершение). Все операторы процедуры обрабатываются при вызове процедуры. Хранимые процедуры используются для группирования операторов T-SQL и любых логических конструкций, необходимых для выполнения задачи. Поскольку хранимые процедуры сохраняются в виде процедурных блоков, они могут использоваться различными пользователями для согласованного повторяемого выполнения одинаковых задач и даже в различных приложениях. Хранимые процедуры также позволяют поддерживать единый подход к управлению задачей, что помогает обеспечивать согласованное и корректное внедрение любых деловых правил.

Использование хранимых процедур может повысить производительность и в других отношениях. Например, использование хранимых процедур для проверки условий сервера может повысить производительность за счет снижения количества данных, которые должны передаваться между клиентом и сервером, и снижения объема обработки, выполняемой на клиентской машине. Для проверки какого-либо условия из хранимой процедуры можно включить в хранимую процедуру условные операторы (например, конструкции IF и WHILE). Логика этой проверки будет обрабатываться на сервере с помощью хранимой процедуры, поэтому вам не потребуется программировать эту логику в самом приложении, а серверу не нужно будет возвращать промежуточные результаты клиенту для проверки данного условия. Вы можете также вызывать хранимые процедуры из сценариев, пакетных заданий и интерактивных командных строк с помощью операторов T-SQL, показанных в примерах далее.

Хранимые процедуры также обеспечивают простой доступ к базе данных для пользователей. Пользователи могут осуществлять доступ к базе данных, не зная деталей архитектуры таблиц и без непосредственного доступа к данным таблиц, – они просто запускают процедуры, которые выполняют требуемые задачи. Тем самым хранимые процедуры помогают обеспечивать соблюдение деловых правил.

Хранимые процедуры могут принимать входные параметры, использовать локальные переменные и возвращать данные. Хранимые процедуры могут возвращать данные с помощью выходных параметров, а также могут возвращать коды завершения, результирующие наборы из операторов SELECT или глобальные курсоры

7.1.2. Программирование хранимых процедур

Оператор **CREATE PROCEDURE** используется для создания хранимых процедур.

CREATE PROCEDURE имя_процедуры [параметры) [опции] AS SQL statement;

Пример

```
CREATE PROCEDURE LateShipments
AS
SELECT   RequiredDate,
         ShippedDate,
         Shippers.CompanyName
FROM     Orders, Shippers
WHERE    ShippedDate    > RequiredDate AND
         Orders.ShipVia = Shippers.ShipperID
GO
```

Хранимые процедуры могут иметь параметры – входные и выходные. Результатом работы хранимой процедуры может быть одно значение, код завершения или список значений. В последнем случае для работы с найденными данными используется курсор.

Чтобы задать входные параметры в хранимой процедуре, укажите список этих параметров с символом @ перед именем каждого параметра, т.е. @имя_параметра, например, @shipperName . Допускается задание до 1024 параметров в хранимой процедуре. Для возврата значения параметра хранимой процедуры в вызывающую программу используйте ключевое слово OUTPUT после имени этого параметра. Чтобы сохранить значение в переменной, которую можно использовать в вызывающей программе, используйте при вызове хранимой процедуры ключевое слово OUTPUT.

Для создания локальных переменных используется ключевое слово DECLARE. При создании локальной переменной вы должны задать для нее имя и тип данных, а также поставить перед именем переменной символ @. При объявлении переменной ей первоначально присваивается значение NULL.

7.2. Курсоры

Запрос к реляционной БД может возвращать несколько строк (записей) данных. Однако приложение может обрабатывать лишь одну запись. В этой ситуации используется концепция курсора, который представляет собой указатель на ряд строк.

Курсор в SQL – это область в памяти БД, которая предназначена для хранения последнего оператора SQL. Если текущий оператор – запрос к БД, в памяти сохраняется строка данных запроса, которая называется текущим значением, или текущей строкой курсора. Указанная область в памяти поименована и доступна для прикладных программ.

Обычно курсоры используются для выбора из БД некоторого подмножества хранимой в ней информации. В каждый момент времени прикладной программой может быть проверена одна строка курсора. Курсоры часто применяются в операторах SQL, встроенных в написанные на языках процедурного типа прикладные программы. Некоторые из них неявно создаются сервером базы данных, в то время как другие определяются программистами.

В соответствии со стандартом SQL при работе с курсорами можно выделить следующие основные действия:

- создание или объявление курсора;
- открытие курсора, т.е. наполнение его данными, которые сохраняются в многоуровневой памяти;
- выборка из курсора и изменение с его помощью строк данных;
- закрытие курсора, после чего он становится недоступным для пользовательских программ;
- освобождение курсора, т.е. удаление курсора как объекта, поскольку его закрытие необязательно освобождает ассоциированную с ним память.

В некоторых случаях применение курсора неизбежно. Однако по возможности этого следует избегать и работать со стандартными командами обработки данных: SELECT, UPDATE, INSERT, DELETE. Помимо того, что курсоры не позволяют проводить операции изменения над всем объемом данных, скорость выполнения операций обработки данных с помощью курсора существенно ниже, чем у стандартных средств SQL.

Для управления курсором bSQL используются следующие команды:

- DECLARE – создание или *объявление курсора* ;
- OPEN – *открытие курсора*, т.е. наполнение его данными;
- FETCH – *выборка из курсора* и *изменение* строк данных с помощью курсора;
- CLOSE – *закрытие курсора* ;
- *DEALLOCATE – освобождение курсора*, т.е. удаление курсора как объекта.

7.2.1. Объявление курсора

В среде MS SQL Server синтаксис команды создания *курсора* имеет вид:

```
DECLARE имя_курсора CURSOR [LOCAL | GLOBAL]
[FORWARD_ONLY | SCROLL]
[STATIC | KEYSET | DYNAMIC | FAST_FORWARD]
[READ_ONLY | SCROLL_LOCKS | OPTIMISTIC]
[TYPE_WARNING]
FOR SELECT_оператор
[FOR UPDATE [OF имя_столбца[,...n]]]
```

При использовании ключевого слова LOCAL будет создан локальный *курсор*, который виден только в пределах создавшего его пакета, триггера, хранимой процедуры или пользовательской функции. По завершении работы пакета, триггера, процедуры или функции *курсор* неявно уничтожается. Чтобы передать содержимое *курсора* за пределы создавшей его конструкции, необходимо присвоить его параметру аргумент OUTPUT.

При использовании ключевого слова GLOBAL, создается глобальный *курсор* ; он существует до закрытия текущего соединения.

При использовании ключевого слова FORWARD_ONLY создается *последовательный курсор* ; *выборку* данных можно осуществлять только в направлении от первой строки к последней.

При использовании ключевого слова SCROLL создается *прокручиваемый курсор* ; обращаться к данным можно в любом порядке и в любом направлении.

При использовании ключевого слова STATIC создается *статический курсор*.

При использовании ключевого слова KEYSET создается ключевой курсор.

При использовании ключевого слова DYNAMIC создается *динамический курсор*.

Если для *курсора* READ_ONLY указать аргумент FAST_FORWARD, то созданный *курсор* будет оптимизирован для быстрого доступа к данным. Этот аргумент не может быть использован совместно с аргументами FORWARD_ONLY иOPTIMISTIC.

В *курсоре*, созданном с указанием аргумента OPTIMISTIC, запрещается *изменение* и *удаление строк*, которые были изменены после *открытия курсора*.

При указании аргумента TYPE_WARNING сервер будет информировать пользователя о неявном изменении типа*курсора*, если он несовместим с запросом SELECT.

7.2.2. Открытие курсора

Для *открытия курсора* и формирования результрующих данных с помощью указанного при создании *курсора* запроса SELECT используется команда:

OPEN {{[GLOBAL]имя_курсора }

|@имя_переменной_курсора}

После открытия курсора происходит выполнение связанного с ним оператора SELECT, выходные данные которого сохраняются в многоуровневой памяти.

7.2.3. Выборка данных из курсора

После *открытия курсора* можно выбрирать его содержимое (результат выполнения соответствующего запроса). Для этого используется команда:

```
FETCH [[NEXT | PRIOR | FIRST | LAST
  | ABSOLUTE {номер_строки
  | @переменная_номера_строки}
  | RELATIVE {номер_строки |
    @переменная_номера_строки}]
  FROM ]{{[GLOBAL ]имя_курсора }|
    @имя_переменной_курсора }
  [INTO @имя_переменной [,...n]]
```

При использовании ключевого слова и FIRST будет возвращена самая первая строка полного результирующего набора курсора, которая становится текущей строкой.

При использовании ключевого слова LAST возвращается самая последняя строка курсора, она становится текущей строкой.

При использовании ключевого слова NEXT возвращается строка, находящаяся в полном результирующем наборе сразу же после текущей. Она становится текущей. По умолчанию команда FETCH использует именно этот способ выборки строк.

При использовании ключевого слова *PRIOR* возвращает строку, находящуюся перед текущей. Она становится текущей.

При использовании ключевого слова ABSOLUTE {номер_строки | @переменная_номера_строки} курсор возвращает строку по ее абсолютному порядковому номеру в полном результирующем наборе курсора. Номер строки можно задать с помощью константы или как имя переменной, в которой хранится номер строки. Переменная должна иметь целочисленный тип данных. Указываются как положительные, так и отрицательные значения. При указании положительного значения строка отсчитывается от начала набора, отрицательного – от конца. Выбранная строка становится текущей. Если указано нулевое значение, строка не возвращается.

При использовании ключевого слова RELATIVE {кол_строки | @переменная_кол_строки} курсор возвращает строку, находящуюся через указанное количество строк после текущей. Если указать отрицательное значение числа строк, то будет

возвращена строка, находящаяся за указанное количество строк перед текущей. При указании нулевого значения возвратится текущая строка. Возвращенная строка становится текущей.

В конструкции INTO @имя_переменной [,...n] задается список переменных, в которых будут сохранены соответствующие значения столбцов возвращаемой строки. Порядок указания переменных должен соответствовать порядку столбцов в *курсоре*, а тип данных переменной – типу данных в столбце *курсора*. Если конструкция INTO не указана, то поведение команды FETCH будет напоминать поведение команды SELECT – данные выводятся на экран.

7.2.4. Изменение и удаление данных

Для выполнения изменений с помощью *курсора* необходимо выполнить команду UPDATE в следующем формате:

```
UPDATE имя_таблицы SET {имя_столбца={
  DEFAULT | NULL | выражение}}[,...n]
  WHERE CURRENT OF {{[GLOBAL] имя_курсора}
  |@имя_переменной_курсора}
```

За одну операцию могут быть изменены несколько столбцов текущей строки *курсора*, но все они должны принадлежать одной таблице.

Для удаления данных посредством *курсора* используется команда DELETE в следующем формате:

```
DELETE имя_таблицы
  WHERE CURRENT OF {{[GLOBAL] имя_курсора}
  |@имя_переменной_курсора}
```

В результате будет удалена строка, установленная текущей в *курсоре*.

7.2.5. Закрытие курсора

CLOSE {имя_курсора | @имя_переменной_курсора}

После *закрытия курсор* становится недоступным для пользователей программы. При *закрытии* снимаются все блокировки, установленные в процессе его работы. *Закрытие* может применяться только к открытым *курсорам*. Закрытый, но не *освобожденный курсор* может быть повторно открыт. Не допускается закрывать неоткрытый*курсор*.

7.2.6. Освобождение курсора

Закрытие курсора необязательно освобождает ассоциированную с ним память. Необходимо явным образом освободить ее с помощью оператора *DEALLOCATE*. После освобождения курсора освобождается память, при этом становится возможным повторное использование имени курсора.

DEALLOCATE { имя_курсора |

@имя_переменной_курсора }

Для контроля достижения конца курсора рекомендуется применять функцию: @@FETCH_STATUS

Функция @@FETCH_STATUS возвращает:

0, если выборка завершилась успешно;

-1, если выборка завершилась неудачно вследствие попытки выборкистроки, находящейся за пределами курсора ;

-2, если выборка завершилась неудачно вследствие попытки обращения к удаленной или измененной строке.

7.3. Пример хранимой процедуры

```
create procedure zapr8 as
declare @yz char (25) /* переменная для названия учебного заведения */
declare @ko char (16) /*переменная для названия категории обучения */
declare @it1 int /* переменная для итога по полю категория обучения */
declare @it2 int /* переменная для итога по полю учебное заведение */
declare @itall int /* переменная для итога по всему запросу */
declare @yz1 char (25) /*переменная для названия учебного заведения*/
/* объявление курсора для двумерного статистического запроса */
declare y cursor for
select vuz.uch_zavedenie, kat_obucheniya, count (*)
from poss join vuz on vuz_k=vuz.cod
        join kat_obuch on kat_obuch_k=kat_obuch.cod
where gp='90'
group by vuz.uch_zavedenie, kat_obuch.kat_obucheniya
select @it2=0 /*начальное значение итога по полю уч. заведение=0*/
select @itall=0 /* начальное значение итога по запросу = 0 */
open y /* открытие курсора */
fetch y into @yz, @ko, @it1
/* считывание первого итога по полю категория обучения */
if (@@fetch_status=-2) begin
print 'Ошибка при выполнении первого FETCH'
close y /* закрытие курсора и останов процедуры в случае ошибки */
```

```
return
end
if (@@fetch_status=-1) begin
print 'Данные не найдены'
close y /*закрытие курсора и останов процедуры в случае отсутствия данных по запросу */
return
end
select @yz1=@yz /*запоминание названия учебного заведения в @yz1*/
print @ko+' -'+str(@it1) /*печать названия категории обучения и итога*/
select @it2=@it2+@it1 /* подсчет итога по полю учебное заведение */
select @itall=@itall+@it1 /* подсчет общего итога по запросу */
/* цикл обработки запроса */
while (@@fetch_status=0)
begin
        fetch y into @yz, @ko, @it1 /* считывание очередного итога по полю категория обучения */
if (@yz1 !=@yz) begin /* если название учебного заведения поменялось, то печать */
        print 'учебное заведение ' + @yz1 + ' - ' +str(@it2) /* старого названия учебного заведения и итога */
select @yz1=@yz/*присвоение нового названия учебного заведения */
select @it2=0 /* новый итог = 0 */
end
if (@@fetch_status=-1) break /* при окончании данных немедленный выход из цикла */
print @ko + ' - ' +str(@it1) /* печать очередных названия категории обучения и итога */
select @it2=@it2+@it1 /* подсчет итога по полю учебное заведение */
```

```
select @itall=@itall+@it1 /* подсчет общего итога по запросу */
end
close y /* закрытие курсора по окончании цикла */
if (@@fetch_status=-2) begin
print 'Ошибка при выполнении FETCH'
return /* останов процедуры в случае ошибки */
end
print 'учебное заведение ' + @yz + ' - ' +str(@it2) /* печать последнего итога по учебному заведению */
print '---------------------------------'
print 'всего по запросу-'+str(@itall) /*печать общего итога по запросу */
deallocate y /* освобождение курсора */
return
```

7.4. Триггеры

Проблема обеспечения целостности БД является одной из наиболее важных при её создании. При определении структуры таблицы (см. лекцию) можно определить статические правила целостности на уровне атрибута или таблицы. Однако, т.к. часто связные данные содержатся в разных таблицах, возникает необходимость поддерживать целостность на уровне нескольких таблиц. Для этого используются триггеры, которые реализуют динамические механизмы поддержания целостности БД. Триггеры - это особый вид хранимых процедур, которые вызываются автоматически при возникновении ситуации, с которой связан триггер. Триггер не может иметь входных параметров и возвращать результат.

В SQL Server существует два вида триггеров, которые различаются по способу его выполнения:

- **AFTER. *Триггер*** выполняется после успешного выполнения вызвавших его команд. Если же команды по какой-либо причине не могут быть успешно завершены, триггер не выполняется. Следует отметить, что изменения данных в результате выполнения запроса пользователя и выполнение триггера осуществляется в теле одной транзакции: если произойдет откат триггера, то будут отклонены и пользовательские изменения. Можно определить несколько AFTER-триггеров для каждой операции (INSERT, UPDATE, DELETE). Если для таблицы предусмотрено выполнение нескольких AFTER-триггеров, то с помощью системной хранимой процедуры sp_settriggerorder можно указать, какой из них будет выполняться первым, а какой последним. По умолчанию в SQL Server все триггеры являются AFTER-триггерами.

- **INSTEAD OF**. Триггер вызывается вместо выполнения команд. В отличие от AFTER-триггера INSTEAD OF-триггер может быть определен как для таблицы, так и для просмотра. Для каждой операции INSERT, UPDATE, DELETE можно определить только один INSTEAD OF-триггер.

Триггеры различают также по типу команд, на которые они реагируют.

Существует три типа триггеров:

- **INSERT TRIGGER** – запускаются при попытке вставки данных с помощью команды INSERT. Использование триггеров INSTEAD OF в операциях INSERT.Если триггер INSTEAD OF определен в операциях INSERT для таблицы или представления, то триггер выполняется вместо инструкции INSERT. Ранние версии SQL Server поддерживают только триггеры AFTER, определенные для инструкции INSERT и других инструкций, изменяющих данные.

- **UPDATE TRIGGER** – запускаются при попытке изменения данных с помощью команды UPDATE.

- **DELETE TRIGGER** – запускаются при попытке удаления данных с помощью команды DELETE.

Конструкции [DELETE] [,] [INSERT] [,] [UPDATE] и FOR | AFTER | INSTEAD OF } { [INSERT] [,] [UPDATE] определяют, на какую команду будет реагировать *триггер*. При его создании должна быть указана хотя бы одна команда. Допускается *создание триггера*, реагирующего на две или на все три команды.

Аргумент WITH APPEND позволяет создавать несколько *триггеров* каждого типа.

При создании триггера с аргументом NOT FOR REPLICATION запрещается его запуск во время выполнения модификации таблиц механизмами репликации.

Конструкция AS sql_оператор[...n] определяет набор SQL- операторов и команд, которые будут выполнены при запуске *триггера*.

Отметим, что внутри триггера не допускается выполнение ряда операций, таких, например, как:

- создание, изменение и удаление базы данных;
- восстановление резервной копии базы данных или журнала транзакций.

Выполнение этих команд не разрешено, так как они не могут быть отменены в случае отката транзакции, в которой выполняется триггер. Это запрещение вряд ли может каким-то образом сказаться на функциональности создаваемых триггеров. Трудно найти такую ситуацию, когда, например, после изменения строки таблицы потребуется выполнить восстановление резервной копии журнала транзакций.

7.4.1. Программирование триггера

При выполнении команд добавления, изменения и удаления записей сервер создает две специальные таблицы: *inserted* и *deleted*. В них содержатся списки строк, которые будут вставлены или удалены по завершении транзакции. Структура таблиц inserted и deleted идентична структуре таблиц, для которой определяется триггер. Для каждого триггера создается свой комплект таблиц inserted и deleted, поэтому никакой другой триггер не сможет получить к ним доступ. В зависимости от типа операции, вызвавшей выполнение триггера, содержимое таблиц inserted и deleted может быть разным:

- команда INSERT – в таблице inserted содержатся все строки, которые пользователь пытается вставить в таблицу; в таблице deleted не будет ни одной строки; после завершения триггера все строки из таблицы inserted переместятся в исходную таблицу;

- команда DELETE – в таблице deleted будут содержаться все строки, которые пользователь попытается удалить; триггер может проверить каждую строку и определить, разрешено ли ее удаление; в таблице inserted не окажется ни одной строки;

- команда UPDATE – при ее выполнении в таблице deleted находятся старые значения строк, которые будут удалены при успешном завершении триггера. Новые значения строк содержатся в таблице inserted. Эти строки добавятся в исходную таблицу после успешного выполнения триггера.

Для получения информации о количестве строк, которое будет изменено при успешном завершении триггера, можно использовать функцию @@ROWCOUNT; она возвращает количество строк, обработанных последней командой. Следует подчеркнуть, что триггер запускается не при попытке изменить конкретную строку, а в момент выполнения команды изменения. Одна такая команда воздействует на множество строк, поэтому триггер должен обрабатывать все эти строки.

Если триггер обнаружил, что из 100 вставляемых, изменяемых или удаляемых строк только одна не удовлетворяет тем или иным условиям, то никакая строка не будет вставлена, изменена или удалена. Такое поведение обусловлено требованиями транзакции – должны быть выполнены либо все модификации, либо ни одной.

Триггер выполняется как неявно определенная транзакция, поэтому внутри триггера допускается применение команд управления транзакциями. В частности, при обнаружении нарушения ограничений целостности для прерывания выполнения триггера и отмены всех изменений, которые пытался выполнить пользователь, необходимо использовать команду ROLLBACK TRANSACTION.

Для получения списка столбцов, измененных при выполнении команд INSERT или UPDATE, вызвавших выполнение триггера, можно использовать функцию COLUMNS_UPDATED(). Она возвращает двоичное число, каждый бит которого, начиная с младшего, соответствует одному столбцу таблицы (в порядке следования столбцов при создании таблицы). Если бит

установлен в значение "1", то соответствующий столбец был изменен. Кроме того, факт изменения столбца определяет и функция UPDATE (имя_столбца).

В теле триггера можно использовать служебные таблицы deleted и inserted.
При выполнении INSERT таблица deleted будет пустая, а в inserted будут содержаться вставленные записи.
При выполнении DELETE таблица inserted будет пустая, а в deleted будут содержаться удаленные записи.
При выполнении UPDATE в deleted содержаться записи до изменения, а в inserted после изменения

Для удаления триггера используется команда

DROP TRIGGER {имя_триггера} [,...n]

7.4.2. Особенности применения триггера

При использовании триггеров необходимо учитывать следующее:

- При внесении изменений в таблицу, триггеры выполняются в последнюю очередь, после проверки всякого рода связей и ограничений.
- Триггер может быть привязан только к одной таблице.
- У одной таблицы может быть много триггеров разных типов.
- Триггеры могут быть на добавление, изменение и удаление (insert, update, delete) в любых вариациях.
- Триггер вызывается на общую команду изменения данных в таблице, т.е. при вызове update для нескольких записей, он не будет выполняться для каждой записи отдельно, а всего один раз.
- Если в триггере изменяются данные по таблице, к которой привязан триггер, то вся группа триггеров по этой таблице выполняется повторно за исключением этого триггера (если в опциях сервера стоит метка "allow nested triggers").
- Писать триггеры надо так, чтобы не была важна последовательность их вызова.

Пример триггера, который выполняет проверку наличия необходимого товара на складе, приведен ниже.

```
CREATE TRIGGER Триггер_ins
ON Сделка FOR INSERT
AS
IF @@ROWCOUNT=1
BEGIN
  IF NOT EXISTS(SELECT *
     FROM inserted
     WHERE inserted.количество<=ALL(SELECT
       Склад.Остаток
    FROM Склад,Сделка
    WHERE Склад.КодТовара=
     Сделка.КодТовара))
    BEGIN
     ROLLBACK TRAN
    PRINT
     'Отмена поставки: товара на складе нет'
    END
END
```

Контрольные вопросы

1. Что такое хранимая процедура?
2. Определите назначение хранимых процедур.
3. Какие виды хранимых процедур Вы знаете?
4. Для чего используется курсор?
5. Определите синтаксис оператора описания курсора.
6. Назовите операторы работы с курсором.
7. Для чего используется оператор FETCH?
8. Назовите отличие триггера от хранимой процедуры.
9. Чем отличается триггер AFTER от триггера INSTEAD OF?
10. Назовите типы триггеров.
11. Для чего при программировании триггера используются системные таблицы inserted и deleted?

ЛЕКЦИЯ 8. ТРАНЗАКЦИИ.

Обеспечение параллелизма при реализации SQL-запросов. Понятие транзакций. Уровни изолированности транзакций. Распределенные транзакции. Методы управления транзакциями.

8.1. Проблемы параллелизма

Современные СУБД являются многопользовательскими системами, т.е. допускают параллельную одновременную работу большого количества пользователей. При этом пользователи, как правило, не знают о том, работают ли с этими данными другие пользователи (или приложения). Для организации такой многопользовательской работы была определена логическая единица работы пользователя с БД – транзакция. Работа СУБД организуется таким образом, что у пользователя складывалось впечатление, что их транзакции выполняются независимо от транзакций других пользователей.

Простейший и очевидный способ обеспечить такую возможность состоит в том, чтобы все поступающие транзакции выстраивать в единую очередь и выполнять строго по очереди. Однако такой способ не подходит по вполне очевидным причинам – теряется преимущество параллельной работы и существенно увеличивается время выполнения запросов пользователей к БД. Таким образом, транзакции необходимо выполнять одновременно, но так, чтобы результат был бы такой же, как если бы транзакции выполнялись по очереди. Основная сложность состоит в том, что если не предпринимать никаких специальных мер, то данные измененные одним пользователем могут быть изменены транзакцией другого пользователя раньше, чем закончится транзакция первого пользователя. В результате, в конце транзакции первый пользователь увидит не результаты своей работы. Т.е. при таком подходе будет нарушена целостность БД при одновременной работе многих пользователей.

8.2. Понятие транзакции и способы управления транзакциями

Понятие транзакции связано с понятием целостности БД, к которой осуществляется одновременный доступ нескольких пользователей. Транзакция – это последовательность операций над БД, рассматриваемых СУБД как единое целое. Либо транзакция успешно выполняется, и СУБД фиксирует (COMMIT) изменения БД, произведенные этой транзакцией, во внешней памяти, либо ни одно из этих изменений никак не отражается на состоянии БД. Понятие транзакции необходимо для поддержания логической целостности БД. Поддержание механизма транзакций является обязательным условием даже однопользовательских СУБД (если, конечно, такая система заслуживает названия СУБД). Но понятие транзакции гораздо более важно в многопользовательских СУБД.

8.2.1. Свойства транзакций

Транзакции обладают следующими свойствами

- Изолированность.
- Атомарность
- Завершенность
- Непротиворечивость

В многопользовательских системах с одной базой данных одновременно могут работать несколько пользователей или прикладных программ. Предельной задачей системы является обеспечение изолированности пользователей, т.е. создание достоверной и надежной иллюзии того, что каждый из пользователей работает с БД в одиночку.

В связи со свойством сохранения целостности БД транзакции являются подходящими единицами изолированности пользователей.

При соблюдении обязательного требования поддержания целостности базы данных возможны следующие уровни изолированности транзакций:

- Первый уровень - отсутствие потерянных изменений. Рассмотрим следующий сценарий совместного выполнения двух транзакций. Транзакция 1 изменяет объект базы данных A. До завершения транзакции 1 транзакция 2 также изменяет объект A. Транзакция 2 завершается оператором ROLLBACK (например, по причине нарушения ограничений целостности). Тогда при повторном чтении объекта A транзакция 1 не видит изменений этого объекта, произведенных ранее. Такая ситуация называется ситуацией потерянных изменений. Естественно, она противоречит

требованию изолированности пользователей. Чтобы избежать такой ситуации в транзакции 1 требуется, чтобы до завершения транзакции 1 никакая другая транзакция не могла изменять объект A. Отсутствие потерянных изменений является минимальным требованием к СУБД по части синхронизации параллельно выполняемых транзакций.

- Второй уровень - отсутствие чтения "грязных данных". Рассмотрим следующий сценарий совместного выполнения транзакций 1 и 2. Транзакция 1 изменяет объект базы данных A. Параллельно с этим транзакция 2 читает объект A. Поскольку операция изменения еще не завершена, транзакция 2 видит несогласованные "грязные" данные (в частности, операция транзакции 1 может быть отвернута при проверке немедленно проверяемого ограничения целостности). Это тоже не соответствует требованию изолированности пользователей (каждый пользователь начинает свою транзакцию при согласованном состоянии базы данных и вправе ожидать видеть согласованные данные). Чтобы избежать ситуации чтения "грязных" данных, до завершения транзакции 1, изменившей объект *A*, никакая другая транзакция не должна читать объект *A* (минимальным требованием является блокировка чтения объекта *A* до завершения операции его изменения в транзакции 1).

- Третий уровень - отсутствие неповторяющихся чтений. Рассмотрим следующий сценарий. Транзакция 1 читает объект базы данных *A*. До завершения транзакции 1 транзакция 2 изменяет объект *A* и успешно завершается оператором COMMIT. Транзакция 1 повторно читает объект *A* и видит его измененное состояние. Чтобы избежать неповторяющихся чтений, до завершения транзакции 1 никакая другая транзакция не должна изменять объект A. В большинстве систем это является максимальным требованием к синхронизации транзакций, хотя, как мы увидим немного позже, отсутствие неповторяющихся чтений еще не гарантирует реальной изолированности пользователей.

- Четвертый уровень – отсутствие фантомов. Рассмотрим следующий сценарий. Транзакция 1 выполняет оператор *A* выборки кортежей отношения *R* с условием выборки S (т.е. выбирается часть кортежей отношения *R*, удовлетворяющих условию *S*). До завершения транзакции 1 транзакция 2 вставляет в отношение *R* новый кортеж *r*, удовлетворяющий условию *S*, и успешно завершается. Транзакция 1 повторно выполняет оператор *A*, и в результате появляется кортеж, который отсутствовал при первом выполнении оператора. Конечно, такая ситуация противоречит идее изолированности транзакций и может возникнуть даже на третьем уровне изолированности транзакций. Чтобы избежать появления кортежей-фантомов, требуется более высокий "логический" уровень синхронизации транзакций

Существует возможность обеспечения разных уровней изолированности для разных транзакций, выполняющихся в одной системе баз данных (в частности, соответствующие операторы предусмотрены в стандарте SQL 2).

Между транзакциями могут существовать следующие виды конфликтов:

- W-W - транзакция 2 пытается изменять объект, измененный не закончившейся транзакцией 1;

- R-W - транзакция 2 пытается изменять объект, прочитанный не закончившейся транзакцией 1;
- W-R - транзакция 2 пытается читать объект, измененный не закончившейся транзакцией 1.

8.2.2. Управление транзакциями

Под управлением транзакциями понимается способность управлять различными операциями над данными, которые выполняются внутри реляционной СУБД. Прежде всего, имеется в виду выполнение операторов INSERT, UPDATE и DELETE. Например, после создания таблицы (выполнения оператора CREATE TABLE) не нужно фиксировать результат: создание таблицы фиксируется в базе данных автоматически. Точно так же с помощью отмены транзакции не удастся восстановить только что удаленную оператором DROP TABLE таблицу.

После успешного выполнения команд, заключенных в тело одной транзакции, немедленного изменения данных не происходит. Для окончательного завершения транзакции существуют так называемые команды управления транзакциями, с помощью которых можно либо сохранить в базе данных все изменения, произошедшие в ходе ее выполнения, либо полностью их отменить.

Существуют три команды, которые используются для управления транзакциями:

- COMMIT – для сохранения изменений;
- ROLLBACK – для отмены изменений;
- SAVEPOINT – для установки особых точек возврата.

После завершения транзакции вся информация о произведенных изменениях хранится либо в специально выделенной оперативной памяти, либо во временной области отката в самой базе данных до тех пор, пока не будет выполнена одна из команд управления транзакциями. Затем все изменения или фиксируются в базе данных, или отбрасываются, а временная область отката освобождается.

Команда COMMIT предназначена для сохранения в базе данных всех изменений, произошедших в ходе выполнения транзакции. Она сохраняет результаты всех операций, которые имели место после выполнения последней команды COMMIT или ROLLBACK..

Команда ROLLBACK предназначена для отмены транзакций, еще не сохраненных в базе данных. Она отменяет только те транзакции, которые были выполнены с момента выдачи последней команды COMMIT или ROLLBACK .

Команда SAVEPOINT (точка сохранения) предназначена для установки в транзакции особых точек, куда в дальнейшем может быть произведен откат (при этом отката всей транзакции не происходит). Команда имеет следующий вид:

SAVEPOINT имя_точки_сохранения

Она служит исключительно для создания точек сохранения среди операторов, предназначенных для изменения данных. Имя точки сохранения в связанной с ней группе транзакций должно быть уникальным.

Для отмены действия группы транзакций, ограниченных точками сохранения, используется команда ROLLBACK со следующим синтаксисом:

ROLLBACK TO имя_точки_сохранения

Поскольку с помощью команды SAVEPOINT крупное число транзакций может быть разбито на меньшие и поэтому более управляемые группы, ее применение является одним из способов управления транзакциями.

8.3. Управление транзакциями в среде MS SQL Server

8.3.1. Определение транзакций

SQL Server предлагает множество средств управления поведением транзакций. Пользователи в основном должны указывать только начало и конец транзакции, используя команды SQL или API (прикладного интерфейса программирования). Транзакция определяется на уровне соединения с базой данных и при закрытии соединения автоматически закрывается. Если пользователь попытается установить соединение снова и продолжить выполнение транзакции, то это ему не удастся. Когда транзакция начинается, все команды, выполненные в соединении, считаются телом одной транзакции, пока не будет достигнут ее конец.

SQL Server поддерживает три вида определения транзакций:

- явное;
- автоматическое;
- подразумеваемое.

По умолчанию SQL Server работает в режиме автоматического начала транзакций, когда каждая команда рассматривается как отдельная транзакция. Если команда выполнена успешно, то ее изменения фиксируются. Если при выполнении команды произошла ошибка, то сделанные изменения отменяются и система возвращается в первоначальное состояние.

Когда пользователю понадобится создать транзакцию, включающую несколько команд, он должен явно указать транзакцию.

Сервер работает только в одном из двух режимов определения транзакций: автоматическом или подразумевающемся. Он не может находиться в режиме исключительно явного определения транзакций. Этот режим работает поверх двух других.

Для установки режима автоматического определения транзакций используется команда:

SET IMPLICIT_TRANSACTIONS OFF

При работе в режиме неявного (подразумевающегося) начала транзакций SQL Server автоматически начинает новую транзакцию, как только завершена предыдущая. Установка режима подразумевающегося определения транзакций выполняется посредством другой команды:

SET IMPLICIT_TRANSACTIONS ON

8.3.2. Описание явных транзакций

Явные транзакции необходимо писывать. Для этого программист должен указать начало и конец транзакции, используя следующие команды:

- начало транзакции: в журнале транзакций фиксируются первоначальные значения изменяемых данных и момент начала транзакции;

BEGIN TRAN[SACTION]

[имя_транзакции |

@имя_переменной_транзакции

[WITH MARK [описание_транзакции]]]

- конец транзакции: если в теле транзакции не было ошибок, то эта команда предписывает серверу зафиксировать все изменения, сделанные в транзакции, после чего в журнале транзакций помечается, что изменения зафиксированы и транзакция завершена;

COMMIT [TRAN[SACTION]

[имя_транзакции |

@имя_переменной_транзакции]]

- создание внутри транзакции точки сохранения: СУБД сохраняет состояние БД в текущей точке и присваивает сохраненному состоянию имя точки сохранения;

SAVE TRAN[SACTION]

{имя_точки_сохранеия |

@имя_переменной_точки_сохранения}

- прерывание транзакции ; когда сервер встречает эту команду, происходит откат транзакции, восстанавливается первоначальное состояние системы и в журнале транзакций отмечается, что транзакция была отменена. Приведенная ниже команда отменяет все изменения, сделанные в БД после оператора BEGIN TRANSACTION или отменяет изменения, сделанные в БД после точки сохранения, возвращая транзакцию к месту, где был выполнен оператор SAVE TRANSACTION.

ROLLBACK [TRAN[SACTION]

[имя_транзакции |

@имя_переменной_транзакции

| имя_точки_сохранения

|@имя_переменной_точки_сохранения]]

Можно использовать следующие функции для контроля за транзакциями:

- Функция @@TRANCOUNT возвращает количество активных транзакций.
- Функция @@NESTLEVEL возвращает уровень вложенности транзакций.

8.3.3. Вложенные транзакции

Вложенными называются транзакции, выполнение которых инициируется из тела уже активной транзакции.

Для создания вложенной транзакции пользователю не нужны какие-либо дополнительные команды. Он просто начинает новую транзакцию, не закрыв предыдущую. Завершение транзакции верхнего уровня откладывается до завершения вложенных транзакций. Если транзакция самого нижнего (вложенного) уровня завершена неудачно и отменена, то все транзакции верхнего уровня, включая транзакцию первого уровня, будут отменены. Кроме того, если несколько транзакций нижнего уровня были завершены успешно (но не зафиксированы), однако на среднем уровне (не самая верхняя транзакция) неудачно завершилась другая транзакция, то в соответствии с требованиями ACID произойдет откат всех транзакций всех уровней, включая успешно завершенные. Только когда все транзакции на всех уровнях завершены успешно, происходит фиксация всех сделанных изменений в результате успешного завершения транзакции верхнего уровня.

Каждая команда COMMIT TRANSACTION работает только с последней начатой транзакцией. При завершении вложенной транзакции команда COMMIT применяется к наиболее "глубокой" вложенной транзакции. Даже если в команде COMMIT TRANSACTION указано имя транзакции более высокого уровня, будет завершена транзакция, начатая последней.

Если команда ROLLBACK TRANSACTION используется на любом уровне вложенности без указания имени транзакции, то откатываются все вложенные транзакции, включая транзакцию самого высокого (верхнего) уровня. В команде ROLLBACK TRANSACTION разрешается указывать только имя самой верхней транзакции. Имена любых вложенных транзакций игнорируются, и попытка их указания приведет к ошибке. Таким образом, при откате транзакции любого уровня вложенности всегда происходит откат всех транзакций. Если же требуется откатить лишь часть транзакций, можно использовать команду SAVE TRANSACTION , с помощью которой создается точка сохранения.

8.3.4. Распределенные транзакции

Распределенные транзакции связаны с использованием данных, расположенных на нескольких ресурсах. Их выполнение затрагивает два или более серверов, которые называются диспетчерами ресурсов. Управление транзакцией должно координироваться между диспетчерами ресурсов компонентом сервера, который называется диспетчером транзакций.

В приложении управление распределенной транзакцией во многом похоже на управление локальной. В конце транзакции приложение запрашивает ее фиксацию или откат. Однако при фиксации распределенной транзакции процесс завершения транзакции реализуется намного сложнее, чтобы свести к минимуму риск сбоя сети, в результате которого одни диспетчеры ресурсов могут фиксировать транзакцию, тогда как другие будут выполнять ее откат. Подходом к управлению распределенной транзакцией является механизма двухфазной фиксации транзакции(2PC), в которой выделяют фазу подготовки и фазу фиксации.

Реализация фазы подготовки заключается в следующем. Когда диспетчер транзакции получает запрос на фиксацию, он отправляет команду подготовки всем диспетчерам ресурсов, занятым в транзакции. Каждый диспетчер ресурсов всемерно обеспечивает устойчивость транзакции, а все буферы, в которых хранятся образы журналов для этой транзакции, записываются на диск. По мере того, как каждый диспетчер ресурсов завершает фазу подготовки, он возвращает диспетчеру транзакций значение успешного или неуспешного завершения подготовки.

Фаза фиксации реализуется по следующему правилу. Если диспетчер транзакций получает значения успешного завершения подготовки от всех диспетчеров ресурсов, то он отправляет команду фиксации каждому диспетчеру ресурсов. После этого диспетчеры ресурсов могут завершить фиксацию. Если все диспетчеры ресурсов сообщают об успешной фиксации, то диспетчер транзакций отправляет уведомление приложению. Если какой-либо диспетчер ресурсов сообщил о неуспешном завершении подготовки, то диспетчер транзакций отправляет команду отката всем диспетчерам ресурсов и сообщает приложению о сбое фиксации.

8.3.5. Уровни изолированности транзакций в SQL Server

Пользователь может управлять уровнем изолированности явных транзакций. Уровень изоляции устанавливается командой:

SET TRANSACTION ISOLATION LEVEL

{ READ UNCOMMITTED

| READ COMMITTED

| REPEATABLE READ

| SERIALIZABLE }

READ UNCOMMITED – незавершенное чтение, или допустимо черновое чтение. Низший уровень изоляции, соответствующий уровню 0. Он гарантирует только физическую целостность данных: если несколько пользователей одновременно изменяют одну и ту же строку, то в окончательном варианте строка будет иметь значение, определенное пользователем, последним изменившим запись. По сути, для транзакции не устанавливается никакой блокировки, которая гарантировала бы целостность данных. Для установки этого уровня используется команда:

SET TRANSACTION ISOLATION

LEVEL READ UNCOMMITTED

READ COMMITTED – завершенное чтение, при котором отсутствует черновое, "грязное" чтение. Тем не менее, в процессе работы одной транзакции другая может быть успешно завершена и сделанные ею изменения зафиксированы. В итоге первая транзакция будет работать с другим набором данных. Это проблема неповторяемого чтения. Данный уровень изоляции установлен в SQL Server по умолчанию и устанавливается посредством команды:

SET TRANSACTION ISOLATION

LEVEL READ COMMITTED

REPEATABLE READ – повторяющееся чтение. Повторное чтение строки возвратит первоначально считанные данные, несмотря на любые обновления, произведенные другими пользователями до завершения транзакции. Тем не менее, на этом уровне изоляции возможно возникновение фантомов. Его установка реализуется командой:

SET TRANSACTION ISOLATION

LEVEL REPEATABLE READ

SERIALIZABLE – сериализуемость. Чтение запрещено до завершения транзакции. Это максимальный уровень изоляции, который обеспечивает полную изоляцию транзакций друг от друга. Он устанавливается командой:

SET TRANSACTION ISOLATION

LEVEL SERIALIZABLE

В каждый момент времени возможен только один уровень изоляции.

8.4. Блокировки

8.4.1. Назначение блокировок

Для управления транзакциями в СУБД используют механизм блокировок. ***Блокировкой*** называется временное ограничение на выполнение некоторых операций обработки данных. *Блокировка* может быть наложена как на отдельную строку таблицы, так и на всю базу данных. *Управлением блокировками* на сервере занимается менеджер блокировок, контролирующий их применение и разрешение конфликтов. *Транзакции* и *блокировки* тесно связаны друг с другом. *Транзакции* накладывают *блокировки* на данные, чтобы обеспечить выполнение требований ACID. Без использования блокировок несколько *транзакций* могли бы изменять одни и те же данные.

Блокировка представляет собой метод управления параллельными процессами, при котором объект БД не может быть модифицирован без ведома транзакции. В этом случае происходит блокирование доступа к объекту со стороны других транзакций. Это позволяет исключить непредсказуемое изменение объекта. Различают два вида блокировки:

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

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

- транзакция, результатом действия которой на строку данных в таблице является ее извлечение, обязана наложить блокировку чтения на эту строку;
- транзакция, предназначенная для модификации строки данных, накладывает на нее блокировку записи;

- если запрашиваемая блокировка на строку отвергается из-за уже имеющейся блокировки, то транзакция переводится в режим ожидания до тех пор, пока блокировка не будет снята;
- блокировка записи сохраняется вплоть до конца выполнения транзакции.

Решение *проблемы параллельной обработки* БД заключается в том, что строки таблиц блокируются, а последующие *транзакции*, модифицирующие эти строки, отвергаются и переводятся в режим ожидания. В связи со свойством сохранения целостности БД транзакции являются подходящими единицами изолированности пользователей. Действительно, если каждый сеанс взаимодействия с базой данных реализуется транзакцией, то пользователь начинает с того, что обращается к согласованному состоянию базы данных – состоянию, в котором она могла бы находиться, даже если бы пользователь работал с ней в одиночку.

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

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

8.4.2. Уровни блокировок

Для решения перечисленных проблем в специально разработанном стандарте определены четыре *уровня блокирования*. Уровень изоляции транзакции определяет, могут ли другие (конкурирующие) транзакции вносить изменения в данные, измененные текущей транзакцией, а также может ли текущая транзакция видеть изменения, произведенные конкурирующими транзакциями, и наоборот. Каждый последующий уровень поддерживает требования предыдущего и налагает дополнительные ограничения:

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

8.4.3. Тупиковые блокировки

"Мертвые", или тупиковые, блокировки характерны для многопользовательских систем. "Мертвая" блокировка возникает, когда две транзакции блокируют два блока данных и для завершения любой из них нужен доступ к данным, заблокированным ранее другой транзакцией. Для завершения каждой транзакции необходимо дождаться, пока блокированная другой транзакцией часть данных будет разблокирована. Но это невозможно, так как вторая транзакция ожидает разблокирования ресурсов, используемых первой. Поэтому в составе каждой СУБД предусмотрен специальный механизм управления такими блокировками.

Без применения специальных механизмов обнаружения и снятия "мертвых" блокировок нормальная работа транзакций будет нарушена. Если в системе установлен бесконечный период ожидания завершения транзакции (а это задано по умолчанию), то при возникновении "мертвой" блокировки для двух транзакций вполне возможно, что, ожидая освобождения заблокированных ресурсов, в тупике окажутся и новые транзакции. Чтобы избежать подобных проблем, в среде MS SQL Server реализован специальный механизм разрешения конфликтов тупикового блокирования. Для этих целей сервер снимает одну из блокировок, вызвавших конфликт, и откатывает инициализировавшую ее транзакцию. При выборе блокировки, которой необходимо пожертвовать, сервер исходит из соображений минимальной стоимости.

Полностью избежать возникновения "мертвых" блокировок нельзя. Хотя сервер и имеет эффективные механизмы снятия таких блокировок, все же при написании приложений следует учитывать вероятность их возникновения и предпринимать все возможные действия для предупреждения этого. "Мертвые" блокировки могут существенно снизить производительность, поскольку системе требуется достаточно много времени для их обнаружения, отката транзакции и повторного ее выполнения. Для минимизации возможности образования "мертвых" блокировок при разработке кода транзакции следует придерживаться следующих правил:

- выполнять действия по обработке данных в постоянном порядке, чтобы не создавать условия для захвата одних и тех же данных;
- избегать взаимодействия с пользователем в теле транзакции;
- минимизировать длительность транзакции и выполнять ее по возможности в одном пакете;
- применять как можно более низкий уровень изоляции.

8.5. Основные задачи администрирования и средства их решения в MS SQL Server

Администратор БД обеспечивает надежную и эффективную работу БД. К основным задачам администрирования БД относятся:

- Инсталляция сервера БД;
- Распределение дисковой памяти и планирование требований системы к параметрам внешней памяти;
- Создание БД
- Управление серверами и сервисами;
- Модификация структуры БД в соответствии с потребностями приложений;
- Управление учетными записями пользователей;
- Отслеживание и оптимизация производительности БД;
- Планирование и выполнение резервного копирования и восстановления БД;
- Архивирование данных.

Для решения перечисленных задач в состав СУДБ, как правило, входят специальные средства (утилиты, менеджеры задач и др.). Эти средства не определяются стандартом языка SQL, поэтому каждый разработчик предлагает свои средства для решения этих задач. Однако существуют общие подходы, на которых основывается решения задач администрирования. Один из них связан с ведением журнала транзакций.

8.5.1. Журнализация

Журнализация является одним из средств обеспечения надежного хранения данных, которое является важнейшим свойством БД. Требование надежного хранения предполагает, в частности, возможность восстановления согласованного состояния базы данных после любого рода аппаратных и программных сбоев. В подавляющем большинстве современных реляционных СУБД такая возможность поддерживается с помощью журнала транзакция, который отражает изменения в БД.

Итак, общей целью журнализации изменений БД является обеспечение возможности восстановления согласованного состояния базы данных после любого сбоя. Поскольку основой поддержания целостного состояния базы данных является механизм транзакций, журнализация и восстановление тесно связаны с понятием транзакции. Общими принципами восстановления являются следующие:

- результаты зафиксированных транзакций должны быть сохранены в восстановленном состоянии базы данных;
- результаты незафиксированных транзакций должны отсутствовать в восстановленном состоянии базы данных.

Журнализация изменений тесно связана с буферизацией страниц БД в оперативной памяти, которая обеспечивает реальный способ достижения удовлетворительной эффективности СУБД. Основным принципом согласованной политики выталкивания буфера журнала и буферов страниц базы данных является то, что запись об изменении объекта БД должна попадать во внешнюю память журнала раньше, чем измененный объект оказывается во внешней памяти БД. Соответствующий протокол журнализации (и управления буферизацией) называется Write Ahead Log (WAL) состоит в том, что если требуется вытолкнуть во внешнюю память измененный объект БД, то перед этим нужно гарантировать выталкивание во внешнюю память журнала записи о его изменении. Т.е., если во внешней памяти БД находится некоторый объект БД, к которому была применена операция модификации, то во внешней памяти журнала обязательно находится запись, соответствующая этой операции. Обратное неверно, т.е. если во внешней памяти журнале содержится запись о некоторой операции изменения объекта БД, то сам измененный объект может отсутствовать во внешней памяти.

8.5.2. Восстановление после сбоя

Различают 2 ситуации сбоя:

- мягкий сбой – при возникновении мягкого сбоя системы утрачивается содержимое оперативной памяти. Восстановление состояния БД состоит в том, что после его завершения БД должна содержать все изменения, произведенные транзакциями, закончившимися к моменту сбоя, и не должна содержать ни одного изменения, произведенного транзакциями, которые к моменту сбоя не закончились. Важно то, что в этом случае состояние БД на внешней памяти не разрушено. Это позволяет сделать процесс восстановления не слишком длительным.
- жесткий сбой приводит к полной или частичной потере содержимого БД на внешней памяти. Как и в случае мягкого сбоя, механизм процесса восстановления тот же: после его завершения БД должна содержать все изменения, произведенные транзакциями, закончившимися к моменту сбоя, и не должна содержать ни одного изменения, произведенного транзакциями, не закончившимися к моменту сбоя. Но такое восстановление возможно только при наличии резервной копии БД.

Восстановление после жесткого сбоя начинается с обратного копирования БД из резервной копии. Затем все закончившиеся транзакции повторно выполняются. Более точно, происходит следующее:

- по журналу в прямом направлении выполняются все операции;
- для транзакций, которые не закончились к моменту сбоя, выполняется откат.

На самом деле, поскольку жесткий сбой не сопровождается утратой буферов оперативной памяти, можно восстановить базу данных до такого уровня, чтобы можно было продолжить даже выполнение незакончившихся транзакций. Но обычно это не делается, потому что восстановление после жесткого сбоя – это достаточно длительный процесс.

8.5.3. Физическая согласованность БД

Для обеспечения физической согласованности используются два основных подхода:

- использование теневого механизма,
- журнализация постраничных изменений базы данных.

Реализация теневого механизма базируется на следующем подходе (рис. 8.1.). При открытии файла таблица отображения номеров его логических блоков в адреса физических блоков внешней памяти считывается в оперативную память. При модификации любого блока файла во внешней памяти выделяется новый блок. При этом текущая таблица отображения (в оперативной памяти) изменяется, а теневая - сохраняется неизменной. Если во время работы с открытым файлом происходит сбой, во внешней памяти автоматически сохраняется состояние файла до его открытия. Для явного восстановления файла достаточно повторно считать в оперативную память теневую таблицу отображения.

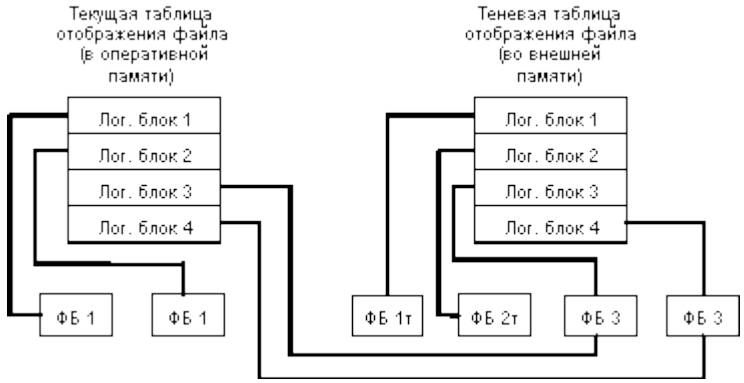

Рис. 8.1. Реализация теневого механизма

Периодически выполняются операции установления точки физической согласованности БД. Для этого все логические операции завершаются, все буфера оперативной памяти, содержимое которых не соответствует содержимому соответствующих страниц внешней памяти, выталкиваются. Теневая таблица отображения файлов базы данных заменяется на текущую (правильнее сказать, текущая таблица отображения записывается на место теневой). Однако такой подход решает проблему восстановления за счет слишком большого расхода внешней памяти. На самом деле, достаточно иметь набор согласованных наборов страниц, каждому из которых может соответствовать свой набор времени.

Для достижения такого более слабого требования наряду с логической журнализацией операций изменения БД производится журнализация постраничных изменений. Первый этап восстановления после мягкого сбоя состоит в постраничном откате незакончившихся логических операций. Подобно тому, как это делается с логическими записями по отношению к транзакциям, последней записью о постраничных изменениях от одной логической операции является запись о конце операции. Для того чтобы распознать, нуждается ли страница внешней памяти базы данных в восстановлении, при выталкивании любой страницы из буфера оперативную память в нее помещается идентификатор последней записи о постраничном изменении этой страницы. Имеются и другие технические подходы, определяемые каждым разработчиком СУБД самостоятельно.

Все более популярным становится поддержание отдельного (короткого) журнала постраничных изменений. Такая техника применяется, например, в известном продукте Informix Online.

8.5.4. Средства администрирования Microsoft SQL Server

MS SQL Server использует журнал с упреждающей записью, который гарантирует, что никакие изменения данных не будут записаны до сохранения на диске записи журнала. Работа механизма упреждающей записи основана на особенностях записи измененных страниц базы данных на диск. MS SQL Server поддерживает буферный кэш, из которого система считывает страницы данных при извлечении необходимых данных. Изменения данных не заносятся непосредственно на диск, а записываются на копии страницы в буферном кэше. Запись измененной страницы данных из буферного кэша на диск называется сбросом страницы на диск. Страница, измененная в кэше, но еще не записанная на диск, называется **грязной** страницей, такие страницы не записываются на диск, пока в базе данных не возникает контрольная точка.

Во время изменения страницы в буфере в кэше журнала строится запись журнала, соответствующая изменению данных. Запись журнала должна быть перенесена на диск до того, как соответствующая «грязная» страница будет записана из буферного кэша на диск. SQL Server реализует алгоритм, который защищает «грязную» страницу от записи на диск до переноса на него соответствующей записи журнала, а запись журнала будет сохранена на диск только после того, как соответствующая транзакция будет зафиксирована.

Создание журнала транзакций происходит во время создания БД по оператору CREATE DATABASE. Опция этой команды LOG ON используется для определения журнала транзакций создаваемой БД. Информация, записывающаяся в журнал транзакций, включает:

- Время начала каждой транзакции;
- Изменения внутри каждой транзакции и информацию для их отката (для этого используются снимки страниц данных до, и после транзакции);
- Информация о распределении памяти для страниц БД (выделении и изъятии экстента);

- Информация о завершении или откате каждой транзакции.

Журнал транзакций поддерживает следующие операции:

- восстановление отдельных транзакций;
- восстановление всех незавершенных транзакций при запуске SQL Server;
- откат восстановленной базы данных, файла, файловой группы или страницы до момента сбоя;
- поддержка репликации транзакций;
- поддержка решений с резервными серверами.

Журнал транзакций является оборачиваемым файлом. Рассмотрим пример. Пусть база данных имеет один физический файл журнала, разделенный на четыре виртуальных файла журнала (рис. 8.2.). При создании базы данных логический файл журнала начинается в начале физического файла журнала. Новые записи журнала добавляются в конце логического журнала и приближаются к концу физического файла журнала. Усечение журнала освобождает любые виртуальные журналы, все записи которых находятся перед минимальным регистрационным номером восстановления в журнале транзакций (MinLSN). MinLSN является в журнале транзакций регистрационным номером самой старой записи, которая необходима для успешного отката на уровне всей базы данных. Журнал транзакций рассматриваемой в данном примере базы данных будет выглядеть примерно так же, как на следующей иллюстрации.

Рис. 8.2. Структура физического файла журнала.

Когда конец логического журнала достигнет конца физического файла журнала, новые записи журнала будут размещаться в начале физического файла журнала (Рис.8.3).

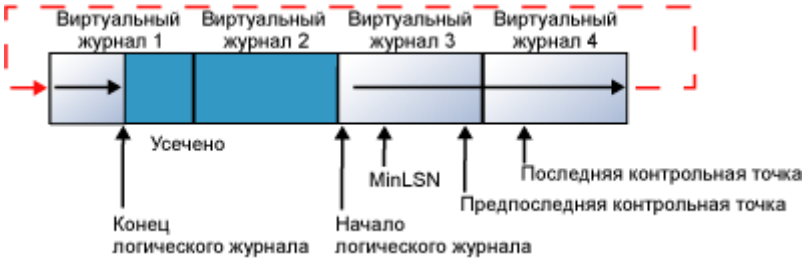

Рис. 8.3. Цикл записи в файл журнала

Этот цикл повторяется бесконечно, пока конец логического журнала не совмещается с началом этого логического журнала. Если старые записи журнала усекаются достаточно часто, так что при этом всегда остается место для новых записей журнала, созданных с новой контрольной точки, журнал постоянно остается незаполненным. Однако, если конец логического журнала совмещается с началом этого логического журнала, происходит одно из двух событий, перечисленных ниже.

- Если для данного журнала применена установка FILEGROWTH и на диске имеется свободное место, файл расширяется на величину, указанную в growth_increment, и новые записи журнала добавляются к этому расширению
- Если установка FILEGROWTH не применяется или диск, на котором размещается файл журнала, имеет меньше свободного места, чем это указано в growth_increment, формируется ошибка 9002.

Если в журнале содержится несколько физических файлов журнала, логический журнал будет продвигаться по всем физическим файлам журнала до тех пор, пока он не вернется на начало первого физического файла журнала.

Логически журнал транзакций SQL Server работает так, как если бы он являлся последовательностью записей в журнале. Каждая запись журнала идентифицируется регистрационным номером транзакции (номер LSN). Каждая новая запись добавляется в логический конец журнала с номером LSN, который больше номера LSN предыдущей записи.

Записи журнала хранятся в той последовательности, в которой они были созданы. Каждая запись журнала содержит идентификатор транзакции, к которой она относится. Все записи журнала, связанные с определенной транзакцией, с помощью обратных указателей связаны в цепочку, которая предназначена для ускорения отката транзакции.

Записи журнала для изменения данных содержат либо выполненную логическую операцию, либо исходный и результирующий образ измененных данных. Исходный образ записи — это копия данных до выполнения операции, а результирующий образ – копия данных после ее выполнения.

Для решения задач администрирования в состав Microsoft SQL Server входит среда SQL Server Management Studio (Рис. 8.4.), которая обладает следующими возможностями:

- поддерживает большинство административных задач для SQL Server;
- реализует единую интегрированную среду для управления компонентами и службами SQL Server: Database Engine, Analysis Services, Reporting Services, Notification Services и выпуске SQL Server Compact, позволяющие выполнять действия немедленно, направлять их в редактор кода или включать эти действия в скрипт для последующего выполнения;
- сохранение и печать XML-файлов плана выполнения и взаимоблокировок, созданных приложением SQL Server Profiler, просмотр их в любое время и отправка для анализа администратору;
- встроенный веб-браузер для быстрого обращения к библиотеке MSDN или получения справки в Интернете;
- новый монитор активности с фильтрацией и автоматическим обновлением;
- встроенные интерфейсы компонента Database Mail.

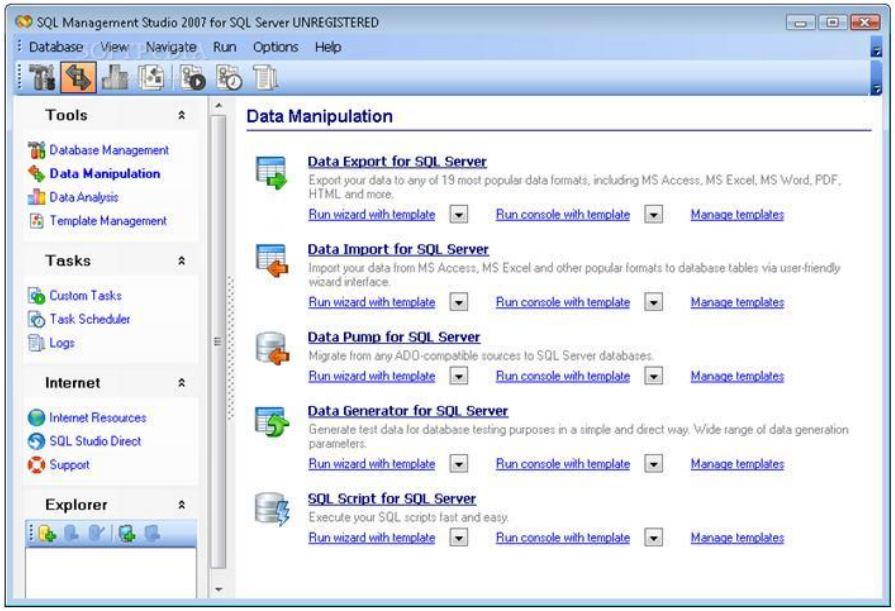

Рис. 8.4. Окно SQL Server Management Studio

В среде SQL Management Studio реализованы следующие задачи администрирования:

- регистрация серверов;
- соединение с экземпляром компонента Database Engine, службами SSAS, службами Службы SSRS, службами SSIS или выпуском SQL Server Compact;
- настройка свойств сервера;
- управление с объектами БД и службами SSAS, такими как кубы, измерения и сборки;
- создание таких объектов, как БД, таблицы, кубы, пользователи БД и имена входа;
- управление файлами и группами файлов;
- присоединение или отсоединение БД;
- запуск средств для работы со сценариями;
- управление безопасностью;
- просмотр системных журналов;
- контроль текущей активности;
- настройка репликации;
- управление полнотекстовыми индексами.

Контрольные вопросы

1. Определите назначение транзакций.
2. Назовите свойства транзакций.
3. Какой оператор позволяет определить начало транзакции?
4. Что такое вложенная транзакция?
5. Опишите особенность реализации распределенной транзакции.
6. Что такое блокировка, и для чего она используется?
7. Что такое тупиковые блокировки?
8. Назовите основные задачи администрирования.
9. Что такое протокол WAL?
10. Для чего используется журнал транзакций?
11. Что такое мягкий сбой?
12. Опишите алгоритм восстановления данных после жесткого сбоя.
13. Дайте характеристику механизмам обеспечения физической согласованности данных.
14. Назовите особенности реализации журнала транзакций в Microsoft SQL Server.
15. Какое программное средство используется для решения задач администрирования в Microsoft SQL Server?

ЛЕКЦИЯ 9. ТЕНДЕНЦИИ РАЗВИТИЯ ТЕХНОЛОГИИ БД

Распределенные БД. Технология OLAP. Хранилища данных. Объектные СУБД - перспективное направление в развитии технологии распределенных баз данных. Объектная модель данных. Архитектура объектной СУБД. Стандарты ODMG.

Технологии БД развивается по нескольким направлениям: развитие БД при использовании в вычислительной сети, развитие моделей данных.

9.1. Распределенные БД

В настоящее время распределенная обработка данных является привычным методом работы с БД. Однако в большинстве случаев распределение касается программных средств обработки данных, сами же данные хранятся централизовано. С развитием сетевых технологий возникает возможность поддерживать не только распределенные приложения, но и распределенные БД. В своей работе К. Дж. Дейт определил распределенную БД как набора узлов (site), связанных коммуникационной сетью, в которой:

- каждый узел – это полноценная СУБД сама по себе;
- узлы взаимодействуют между собой таким образом, что пользователь любого из них может получить доступ к любым данным в сети так, как будто они находятся на его собственном узле.

Из этого определения следует, что так называемая *распределенная база данных* в действительности представляет собой *виртуальную* БД, компоненты которой физически хранятся в нескольких различных *реальных* БД на нескольких различных узлах.

К. Дж. Дейт определил 12 свойств или качеств идеальной распределенной БД:

- Локальная автономия (local autonomy)
- Независимость узлов (no reliance on central site)
- Непрерывные операции (continuous operation)
- Прозрачность расположения (location independence)
- Прозрачная фрагментация (fragmentation independence)
- Прозрачное тиражирование (replication independence)
- Обработка распределенных запросов (distributed query processing)
- Обработка распределенных транзакций (distributed transaction processing)

- Независимость от оборудования (hardware independence)
- Независимость от операционных систем (operationg system independence)
- Прозрачность сети (network independence)
- Независимость от СУБД (database independence)

Рассмотрим подробнее выделенные свойства.

9.1.1. Локальная автономия

Это свойство означает, что управление данными на каждом из узлов распределенной системы выполняется локально. База данных, расположенная на одном из узлов, является неотъемлемым компонентом распределенной системы, но в то же время она функционирует как полноценная локальная база данных, т.е. управление ею выполняется локально и независимо от других узлов системы.

9.1.2. Независимость от центрального узла

В идеальной распределенной БД все узлы равноправны и независимы, а расположенные на них БД являются равноправными поставщиками данных в общее пространство данных. БД на каждом из узлов самодостаточна – она включает полный собственный словарь данных и полностью защищена от несанкционированного доступа.

9.1.3. Непрерывные операции

Это свойство можно определить как обеспечение непрерывного доступа к данным в рамках распределенной БД, которое не зависит от расположения данных и выполняемых операций.

9.1.4. Прозрачность расположения

Это свойство означает, что пользователь не должен знать о реальном (физическом) размещении данных в узлах распределенной БД. Все операции над данными выполняются без учета их местонахождения. Распределение запросов к реальным БД осуществляется средствами системы. Таким образом. Запрос к распределенной БД ничем не отличается от запроса к обычной (локальной) БД.

9.1.5. Прозрачная фрагментация

Это свойство предоставляет возможность распределенного размещения данных, которые логически представляющих собой единое целое. Различают фрагментацию двух типов: горизонтальную и вертикальную. В первом случае хранение строк одной таблицы может быть выполнено на различных узлах (фактически, хранение строк одной

логической таблицы в нескольких идентичных физических таблицах на различных узлах). Во втором случае столбцы логической таблицы располагаются по нескольким узлам.

9.1.6. Прозрачность тиражирования

Тиражирование данных – это асинхронный (в общем случае) процесс переноса изменений объектов исходной БД в базы, расположенные на других узлах распределенной системы. Таким образом, прозрачность тиражирования обеспечивает возможность переноса изменений между БД средствами распределенной СУБД.

9.1.7. Обработка распределенных запросов

Это свойство распределенной БД определяется как возможность выполнения операций выборки над распределенной БД, сформулированных в рамках обычного запроса на языке SQL. То есть операцию выборки из распределенной БД можно сформулировать с помощью тех же языковых средств, что и операцию над локальной базой данных.

9.1.8. Обработка распределенных транзакций

Это свойство распределенной БД гарантирует выполнение операций обновления распределенной базы данных, которые не разрушают целостность и согласованность данных. Эта цель достигается применением двухфазового или двухфазного протокола фиксации транзакций.

9.1.9. Независимость от оборудования

Это свойство является скорее пожеланием, оно означает, что в качестве узлов распределенной системы могут выступать компьютеры любых моделей и производителей – от мэйнфреймов до персональных компьютеров.

9.1.10. Независимость от операционных систем

Это свойство является следствием предыдущего, и тоже может быть отнесено, скорее, к пожеланиям. Оно означает возможность поддержания в рамках распределенной СУБД разнообразных операционных систем, управляющих узлами распределенной системы.

9.1.11. Прозрачность сети

Доступ к компонентам распределенной БД может осуществляться по сети. Спектр поддерживаемых конкретной СУБД сетевых протоколов не должен быть ограничением системы с распределенными БД. Данное качество формулируется максимально широко и предполагает поддержку любых сетевых протоколов.

9.1.12. Независимость от СУБД

Это качество означает, что в распределенной системе могут мирно сосуществовать СУБД различных производителей, и возможны операции поиска и обновления в базах данных различных моделей и форматов.

9.2. Технология OLAP и хранилища данных

Технология OLAP (Online Analytical Processing) предназначена для решения задач оперативного анализа данных. OLAP предоставляет удобные быстродействующие средства доступа, просмотра и анализа информации. В 1993 г. К. Дж. Кодд сформулировал 12 определяющих принципов OLAP. Позднее на их основе были разработаны критерии, позволяющие определить принадлежность к OLAP-системе. Они были оформлены в виде теста FASMI.

9.2.1. Тест FASMI

- Fast (Быстрый) – анализ должен производиться одинаково быстро по всем аспектам информации. Приемлемое время отклика должно составлять не более 5 с.

- Analysis (Анализ) – должна быть возможность осуществлять основные типы числового и статистического анализа, предопределенного разработчиком приложения или произвольно определяемого пользователем.

- Shared (Разделяемой) – множество пользователей должно иметь доступ к данным, при этом необходимо контролировать доступ к конфиденциальной информации.

- Multidimensional (Многомерной) – это основная, наиболее существенная характеристика OLAP, которая предполагает использование многомерной модели данных в OLAP-системах.

- Information (Информации) – приложение должно иметь возможность обращаться к любой нужной информации, независимо от ее объема и места хранения.

9.2.2. Принципы построения многомерной модели данных

В основе модели данных в OLAP-системах лежит многомерное представление – многомерный куб (Cubes). Осями многомерной системы координат служат основные атрибуты анализируемых данных (рис. 9.1). На пересечениях осей-измерений (Dimensions) находятся данные, количественно характеризующие анализируемые данные, они называются меры (Measures). Такое многомерное представление позволяет работать с данными как с многомерным массивом, благодаря чему обеспечиваются одинаково быстрые вычисления суммарных показателей и различные многомерные преобразования по любому из измерений. Например, при анализе продаж анализируемыми данными (мерами) могут быть определены объемы продаж, а измерения фиксируют дату продаж, место продаж, вид товара. Это также могут быть объемы продаж в штуках или в денежном выражении, остатки на складе, издержки и т. п. Пользователь может анализировать данные и получать сводные (например, по годам) или, наоборот, детальные (по неделям) сведения и осуществлять другие виды анализа в соответствии со своими потребностями.

Данные, хранящиеся в OLAP-системах, представляют собой данные разного вида и назначения:

1. детальные данные – данные, которые переносятся, как правило, непосредственно из OLTP-подсистем. Они соответствуют элементарным событиям, фиксируемым в OLTP-системах. Подразделяются на:
   - измерения – это наборы данных, необходимые для описания событий (например, товар, продавец, покупатель, магазин, и др.);
   - факты – это данные, отражающие сущность события (например, количество проданного товара, сумма продаж, и др.);
2. агрегированные (обобщенные) данные – это данные, получаемые на основании детальных путем суммирования по определенным измерениям;
3. метаданные – это данные о данных, содержащихся в OLTP-системе. Они могут описывать:
   - объекты предметной области, информация о которых содержится в OLTP-системе;
   - категории пользователей, использующих данные в OLTP-системе;
   - места и способы хранения данных;
   - действия, выполняемые над данными;
   - время выполнения различных действий над данными;
   - причины выполнения различных действий над данными.

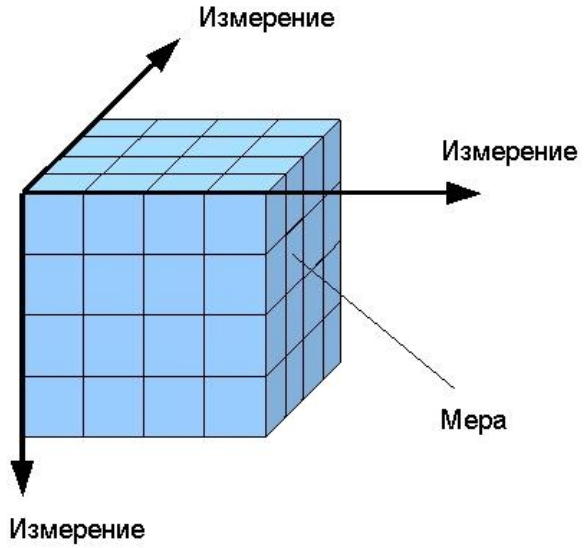

Рис. 9.1. Многомерный куб

Оси куба представляют собой **измерения**, по которым откладывают параметры, относящиеся к анализируемой предметной области, например, названия товаров и названия месяцев года. На пересечении осей измерений располагаются **данные**, количественно характеризующие анализируемые факты – **меры**, например, объемы продаж, выраженные в единицах продукции.

Значения, "откладываемые" вдоль измерений, называются членами или метками (members). Метки используются как для "разрезания" куба, так и для ограничения (фильтрации) выбираемых данных — когда в измерении, остающемся "неразрезанным", нас интересуют не все значения, а их подмножество, например три города из нескольких десятков. Значения меток отображаются в двумерном представлении куба как заголовки строк и столбцов. Важно отметить, что каждое из измерений OLAP-куба может быть представлено в виде иерархической структуры. Например, измерение "Регион" может иметь следующие уровни иерархии: "страна – федеральный округ – область – город – район". Допускается несколько уровней иерархического представления для измерения, например, измерение "время" может иметь иерархию представления "год – квартал – месяц – день" и иерархию представления "год – неделя – день".

9.2.3. Операции, выполняемые над гиперкубом

Над гиперкубом могут выполняться следующие операции:

1. Срез (рис.9.2) - формируется подмножество многомерного массива данных, соответствующее единственному значению одного или нескольких элементов измерений, не входящих в это подмножество.

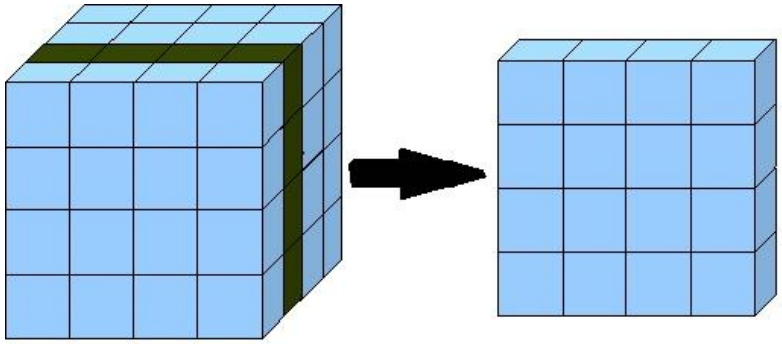

Рис. 9.2. Операция срез

2. Вращение (рис. 9.3.) - изменение расположения измерений, представленных в отчете или на отображаемой странице. Например, операция вращения может заключаться в перестановке местами строк и столбцов таблицы. Кроме того, вращением куба данных является перемещение внетабличных измерений на место измерений, представленных на отображаемой странице, и наоборот.
3. Консолидация (Рис. 9.4) и детализация (Рис. 9.5) – операции, которые определяют переход от детального представления данных к агрегированному и наоборот, соответственно. Направление детализации (обобщения) может быть задано как по иерархии отдельных измерений, так и согласно прочим отношениям, установленным в рамках измерений или между измерениями

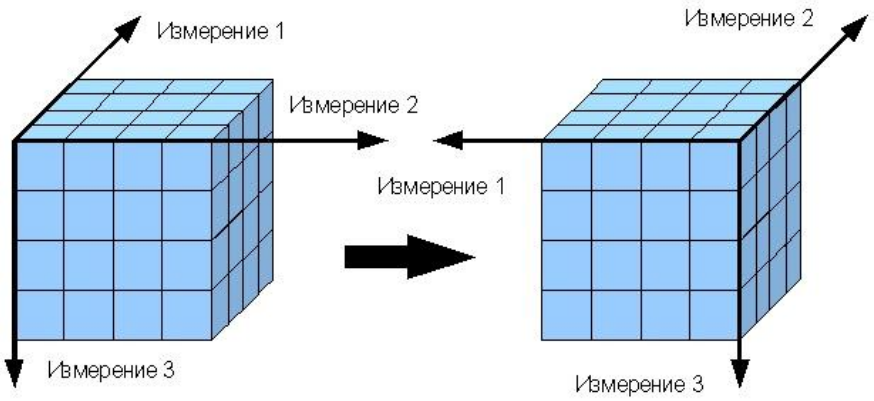

Рис. 9.3. Операция вращение

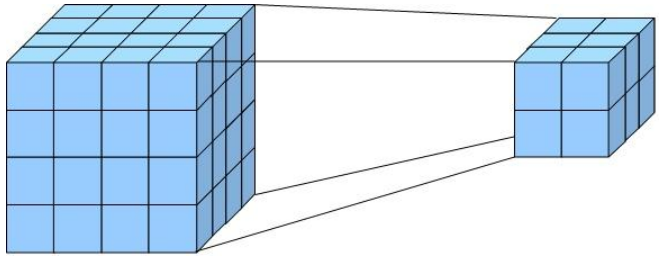

Рис. 9.4. Операция консолидация

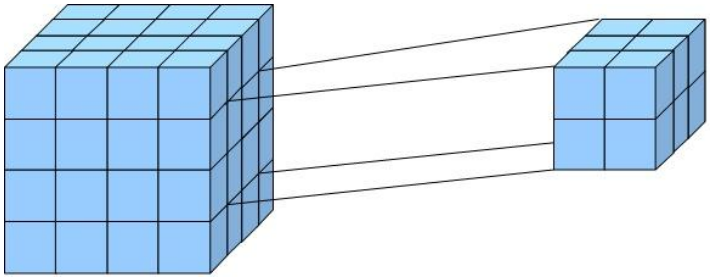

Рис. 9.5. Операция детализация

9.2.4. Понятие хранилища данных

Технология хранилища данных (ХД) предназначена для хранения и анализа больших объемов данных с целью дальнейшего обнаружения в них скрытых закономерностей и, наряду с технологией Data Mining, входит в понятие "предсказательная аналитика". Data Mining, в свою очередь, изучает процесс нахождения новых, действительных и потенциально полезных знаний в базах данных.

ХД – предметно-ориентированный, интегрированный, редко меняющийся, поддерживающий хронологию набор данных, организованный для целей поддержки принятия решений. Предметная ориентация означает, что ХД интегрируют информацию, отражающую различные точки зрения на предметную область. Интеграция предполагает, что данные, хранящиеся в ХД, приводятся к единому формату. Поддержка хронологии означает, что все данные в ХД соответствуют последовательным интервалам времени.

Основная проблематика при создании ХД заключается в следующем:

1. интеграция разнородных данных. Данные в ХД поступают из разнородных OLTP-систем, которые физически могут быть расположены на различных узлах сети. При проектировании и разработке ХД необходимо решать задачу интеграции различных программных платформ хранения;

2. эффективное хранение и обработка больших объемов данных. Построение ХД предполагает накопление данных за значительные периоды времени, что ведет к постоянному росту объемов дисковой памяти, а также росту объема оперативной памяти, требующейся для обработки этих данных. При возрастании объемов данных этот рост нелинеен;

3. организация многоуровневых справочников метаданных. Конечным пользователям СППР необходимы метаданные, описывающие структуру хранящихся в ХД данных, а также инструменты их визуализации;

4. обеспечение информационной безопасности ХД. Сводная информация о деятельности компании, как правило, относится к коммерческой тайне и подлежит защите; кроме того, в ХД могут содержаться персональные данные клиентов и сотрудников, которые также необходимо защищать. Для выполнения этой функции должна быть разработана политика безопасности ХД и связанной с ним инфраструктуры, а также реализованы предусмотренные в политике организационные и программно-технические мероприятия по защите информации.

В архитектуре ХД (рис. 9.6) выделяют следующие уровни:

- уровень оперативного источника данных, на котором происходит интеграция данных из различных источников;
- уровень хранилища данных, на которым происходит агрегированиие данных, формирование репозитария метаданных;

- уровень анализа, на котором происходит аналитическая обработка многомерных данных и визуализация результатов.

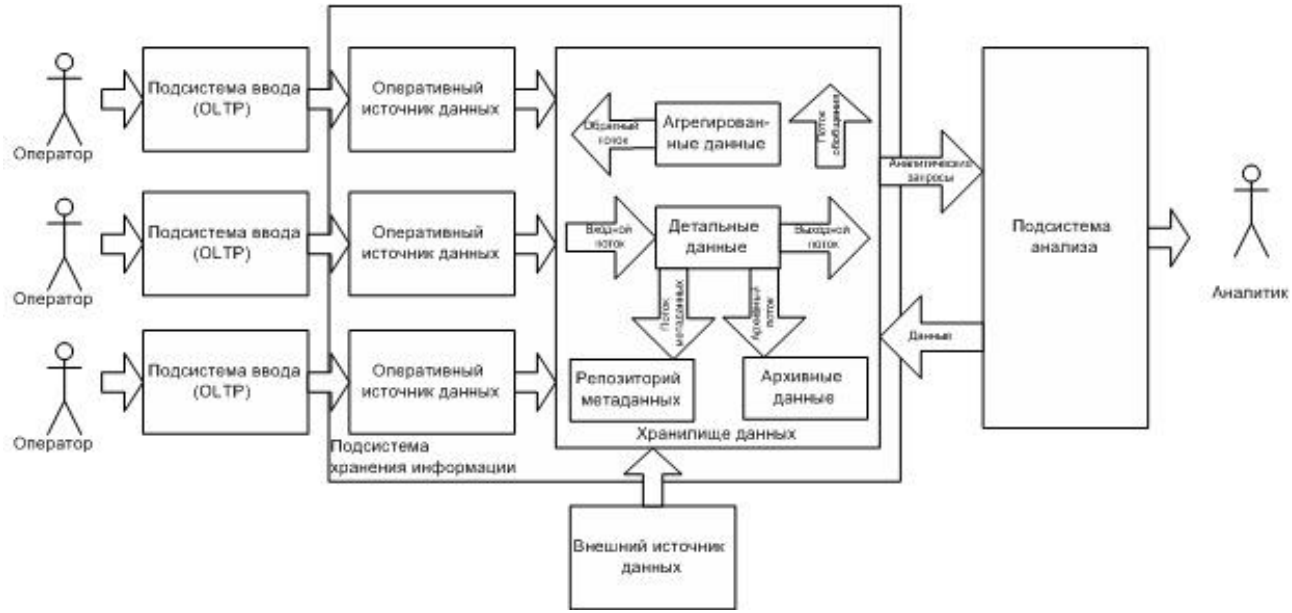

Рис. 9.6. Архитектура хранилищ данных

9.3. Объектная модель данных

Направление объектно-ориентированных баз данных (ООБД) возникло сравнительно давно. Публикации появлялись уже в середине 1980-х. Объектная технология является важной областью в сфере разработки программного обеспечения. Поэтому развитие моделей БД в направлении объектного подхода является актуальной. Возникновение направления ООБД определяется, прежде всего, потребностями практики: необходимостью разработки сложных информационных прикладных систем, для которых технология реляционных СУБД не была вполне подходящей. Хотя существует большое количество экспериментальных проектов (и даже коммерческих систем), в настоящее время отсутствует общепринятая объектно-ориентированная модель данных.

9.3.1. Проблемы разработки ООБД

В классической постановке объектно-ориентированный подход базируется на следующих концепциях:

- объекта и идентификатора объекта;
- атрибутов и методов;
- классов;
- иерархии и наследования классов.

В прикладных информационных системах, основывавшихся на традиционных реляционных БД, существует принципиальное разделение структурной и поведенческой частей. Структурная часть системы поддерживается СУБД, а поведенческая часть реализуется отдельно. В среде ООБД проектирование, разработка и сопровождение прикладной системы становится процессом, в котором должны быть интегрированы структурный и поведенческий аспекты информационных объектов предметной области. Для этого нужны специальные языки, позволяющие определять объекты предметной области в традиционном для объектно-ориентированного подхода понимании. Несмотря на то, что между языками программирования и теорией управления базами данных, бесспорно, имеется много общего, существует очень важное различие. Оно заключается в том, что прикладная программа предназначена для решения некоторых конкретных задач. В то время как БД предназначена для решения целого ряда различных задач, формулировка которых может быть даже не известна в момент создания базы данных.

Специфика применения объектно-ориентированного подхода для организации и управления БД потребовала уточнения в трактовке классических концепций и их расширения. Это связано, прежде всего, с тем, что объекты БД хранятся во внешней памяти, а не в оперативной. Требуется обеспечить ассоциативный доступ к объектам, согласованное состояние ООБД в условиях многопользовательского доступа и других особенностей, свойственных базам данных. Основные трудности, связанные с

созданием объектно-ориентированной модели БД, определяются тем, что пока не найден такой развитый математический аппарат, на который могла бы опираться общая объектно-ориентированная модель данных.

В последнее широкий интерес представляют особый тип объектов – документы XML. Проблема хранения таких документов в БД и выполнения запросов над такими объектами приобретает важное практическое значение и имеет некоторые практические результаты.

9.3.2. Стандарт ODMG

В 1991 г. был образован консорциум ODMG (первоначально означала *Object Database Management Group* , но впоследствии приобрела более широкую трактовку – *Object Data Management Group*). Он выработал предложения по архитектуре ООБД (рис. 9.7), в которой выделены ODL (*Object Definition Language – язык определения объектов)*, OQL (*Object Query Language – язык объектных запросов)* и OML (*Object Manipulation Language – язык манипулирования объектами)*.

Центральной в архитектуре является *модель данных*, представляющая организационную структуру, в которой сохраняются все данные, управляемые ООСУБД. Язык определения объектов, язык запросов и языки манипулирования разработаны таким образом, что все их возможности опираются на модель данных. Архитектура допускает существование разнообразных реализационных структур для хранения моделируемых данных, но важным требованием является то, что все программные библиотеки и все поддерживающие инструментальные средства обеспечиваются в объектно-ориентированных рамках и должны сохраняться в согласовании с данными.

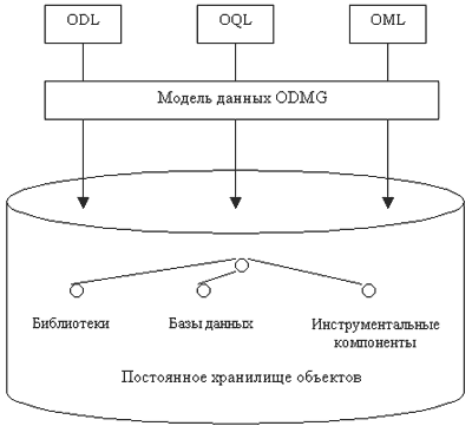

Рис. 9.7. Архитектура ООБД

Основными компонентами архитектуры являются следующие.

- *Объектная модель данных.* Все данные, сохраняемые ООСУБД, структурируются в терминах конструкций модели данных. В модели данных определяется точная семантика всех понятий.

- *Язык определения данных (ODL).* Схемы баз данных описываются в терминах языка ODL , в котором конструкции модели данных конкретизируются в форме языка определения. ODL позволяет описывать схему в виде набора интерфейсов объектных типов, что включает описание свойств типов и взаимосвязей между ними, а также имен операций и их параметров. ODL не является полным языком программирования; реализация типов должна быть выполнена на одном из языков категории OML . Кроме того, ODL является *виртуальным* языком в том смысле, что в стандарте ODMG не требуется его реализация в программных продуктах ООСУБД, которые считаются соответствующими стандарту. Допускается поддержка этими продуктами эквивалентных языков определения, включающих все возможности ODL , но адаптированных под особенности конкретной системы. Тем не менее, наличие спецификации языка ODL в стандарте ODMG является важным, поскольку в языке конкретизируются свойства модели данных.

- *Язык объектных запросов (ODL).* Язык имеет синтаксис, похожий на синтаксис языка SQL, но опирается на семантику объектной модели ODMG . В стандарте допускается прямое использование OQL и его встраивание в один из языков категории OML .

- *Языки манипулирования объектами (OML).* Для программирования реализаций операций и приложений требуется наличия объектно-ориентированного языка программирования. OML представляется собой интегрирование языка программирования с моделью ODMG; это интегрирование производится за счет определенных в стандарте правил *языкового связывания (language binding)*. Дело в том, что в самих языках программирования, естественно, не поддерживается стабильность объектов. Чтобы разрешить программам на этих языках обращаться к хранимым данным, языки должны быть расширены дополнительными конструкциями или библиотечными элементами. Эту возможность и обеспечивает языковое связывание.

- *Постоянное хранилище объектов.* Логическая организация хранилища данных любой ООСУБД, совместимой со стандартном ODMG, должна основываться на модели данных ODMG. Физическая организация у разных ООСУБД может различаться. Она должна обеспечивать эффективные структуры данных для хранения иерархии типов и объектов, являющихся экземплярами этих типов. Иерархия типов связана не только с данными, но и с различными библиотеками и компонентами инструментальных средств, которые поддерживают разработку приложений. Так что в ООСУБД, совместимой со стандартом ODMG, хранилище представляет собой интегрированную систему, где согласованным образом сохраняются данные и программный код.

- *Инструментальные средства и библиотеки.* Инструментальные средства, поддерживающие, например, разработку пользовательских приложений и их графических интерфейсов, программируются на одном из OML и сохраняются как часть

иерархии классов. Библиотеки функций доступа, арифметических функций и т.д. также сохраняются в иерархии типов и являются единообразно доступными из программного кода разработчика приложения. Ассортимент инструментальных средств и библиотек в стандарте не определяется.

Контрольные вопросы

1. Перечислите свойства распределенных БД.
2. Что такое прозрачная фрагментация?
3. Дайте характеристику процесса обработки распределенной транзакции.
4. Определите назначение теста FASMI.
5. Определите специфику многомерной модели данных.
6. Что такое метаданные многомерной модели?
7. Определите операции работы с многомерной моделью.
8. Что такое хранилище данных?
9. Назовите уровни архитектуры хранилища данных.
10. Дайте характеристику компонентов архитектуры ООБД.