МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

ЛЕКЦИОННЫЕ МАТЕРИАЛЫ

Технологии хранения в системах кибербезопасности

(наименование дисциплины (модуля) в соответствии с учебным планом)

Уровень	бакалавриат
	(бакалавриат, магистратура, специалитет)
Форма обучения	очная
	(очная, очно-заочная, заочная)
Направление(-я) подготовки	10.05.04 Информационно-аналитические системы безопасности
	(код(-ы) и наименование(-я))
Институт	Кибербезопасности и цифровых технологий (ИКБ)
	(полное и краткое наименование)
Кафедра	КБ-2 «Прикладные информационные технологии»
	(полное и краткое наименование кафедры, реализующей дисциплину (модуль))
Лектор	к.т.н., Селин Андрей Александрович
	(сокращенно – ученая степень, ученое звание; полностью – ФИО)

Используются в данной редакции с учебного года 2024/2025

(учебный год цифрами)

Проверено и согласовано «____» _____2024 г. _____ А.А. Бакаев

(подпись директора Института/Филиала с расшифровкой)

Москва 2024 г.

Технологии хранения в системах кибербезопасности

2024 год

Учебные вопросы лекции:

1. Файловые системы и системы с базами данных
2. Архитектура системы баз данных

Введение

С начала развития **вычислительной техники** (**ВТ**) прослеживаются два основных направления ее использования:

1. Применение ВТ для выполнения численных расчетов, которые слишком долго или вообще невозможно производить вручную.

2. Использование средств ВТ в **автоматизированных информационных системах (АИС)**, предназначенных для поддержки надежного хранения информации, выполнения специфических для каждого приложения преобразований информации, вычислений и др.

Основные ресурсы, в том числе и вычислительные, изначально находились под контролем военных, и, соответственно, все исследования носили закрытый характер. ЭВМ обладали ограниченными возможностями по отношению к хранению данных. Развитие аппаратных и программных средств управления внешней памятью диктовалось потребностями **информационных систем (ИС)**, для построения которых требовалась возможность надежного и долговременного хранения больших объемов данных, а также обеспечение достаточно быстрого доступа к этим данным. По мере развития ИС возрастали и их потребности, что привело к необходимости наличия отдельного информационного компонента – **базы данных (БД)** и отдельного управляющего компонента – **системы управления БД (СУБД)**.

1. Файловые системы и системы с базами данных

С точки зрения прикладной программы **файл** – это именованная область внешней памяти, в которую можно записать или из которой можно считать данные. Система управления файлами берет на себя функции распределения внешней памяти, отображения имен файлов в соответствующие адреса внешней памяти и обеспечения доступа к данным.

В современных файловых системах обеспечивается многоуровневое именование файлов за счет наличия во внешней памяти **каталогов** – дополнительных файлов со специальной структурой. Поддержка многоуровневой схемы именования файлов обеспечивает простую и удобную схему логической классификации файлов и генерации их имен.

Файловая система является общим хранилищем файлов, принадлежащих разным пользователям. Для авторизации доступа к файлам используются:

1. **Мандатный способ** – каждый пользователь имеет или не имеет отдельный мандат для работы с каждым файлом. В общем виде подход состоит в том, что по отношению к каждому зарегистрированному пользователю для каждого файла указываются действия, которые разрешены или запрещены данному пользователю.
2. **Дискреционный способ** – помечается, какие действия с файлом могут производить пользователи различных групп. Для каждого файла контролируется возможность исполнения трех действий: чтение, запись и выполнение.

Файловые системы и системы с базами данных (продолжение)

Если операционная система поддерживает многопользовательский режим, может возникнуть ситуация, в которой два или более пользователей пытаются работать с одним и тем же файлом одновременно. Если эти пользователи собираются только читать файл, ничего страшного не произойдет. Однако, если хотя бы один из них будет изменять файл, то для корректной работы требуется взаимная синхронизация.

Обычно, в файловых системах применяется следующий подход:

– в операции открытия файла помимо прочих параметров указывается режим работы (чтение или изменение). Если к моменту выполнения операции открытия файла от имени некоторого процесса ***A*** файл уже находится в открытом состоянии от имени другого процесса ***B***, причем режим открытия несовместим (совместимы только режимы чтения), то в зависимости от особенностей системы процессу ***A*** сообщается о невозможности открытия файла или процесс ***A*** блокируется до тех пор, пока процесс ***B*** не выполнит операцию закрытия файла.

Файлы, содержащие тексты программ, используются как входные файлы компиляторов (чтобы правильно воспринять текст программы, компилятор должен понимать логическую структуру текстового файла), которые, в свою очередь, формируют файлы, содержащие объектные модули.

Файловые системы и системы с базами данных (окончание)

С точки зрения файловой системы объектные файлы обладают очень простой структурой – последовательность записей или байтов. Система программирования накладывает на такую структуру более сложную и специфичную для этой системы структуру объектного модуля.

Ситуация аналогична и в других случаях: например, при использовании файлов, содержащих графическую, аудио- и видеоинформацию.

Таким образом, файловые системы обеспечивают хранение слабоструктурированной информации, оставляя дальнейшую структуризацию прикладным программам. В некоторых случаях использования файлов это даже хорошо, потому что при разработке любой новой прикладной системы, опираясь на простые, стандартные и сравнительно дешевые средства файловой системы, можно реализовать те структуры хранения, которые наиболее точно соответствуют специфике предметной области.

Потребности информационных систем

ИС ориентированы на хранение, выборку и модификацию информации, структура которой достаточно сложна. На начальном этапе использования вычислительной техники проблемы структуризации данных решались индивидуально для каждой ИС. Так появлялись надстройки над файловыми системами – библиотеки программ, реализующие сложные методы хранения данных. Однако, невозможно обойтись только библиотеками программ. Предположим, требуется реализовать ИС, поддерживающую учет сотрудников некоторой организации. Система должна выполнять следующие функции:

– выдавать списки сотрудников по отделам;

– поддерживать возможность перевода сотрудника из одного отдела в другой;

– обеспечивать возможность приема на работу новых сотрудников и увольнения работающих.

Помимо этого, для каждого отдела должна поддерживаться возможность получения имени руководителя отдела, общей численности отдела, суммарного и среднего размеров зарплаты сотрудников отдела. Для каждого сотрудника должна поддерживаться возможность получения полного имени по номеру удостоверения, информации о размере его зарплаты.

Потребности информационных систем (продолжение)

Предположим, что требуется реализовать данную ИС на основе файловой системы с использованием одного файла "Сотрудники". Минимальной информационной единицей является сотрудник. Следовательно, в файле "Сотрудники" должна содержаться одна запись для каждого сотрудника. Для удовлетворения этих требований запись о сотруднике должна содержать поля:

– номер удостоверения (Сотр_Номер);

– полное имя (Сотр_Имя);

– размер зарплаты (Сотр_Зарп);

– номер отдела (Сотр_Отд_ном) ;

– имя руководителя отдела (Сотр_Рук_отд).

Для эффективного функционирования ИС требуется обеспечить доступ к файлу "Сотрудники" по уникальному (недублируемому) ключу *Сотр_Номер*. Кроме того, должна обеспечиваться возможность эффективного выбора записей о сотрудниках и отделах, т.е. доступ по неуникальным ключам *Сотр_Имя* и *Сотр_Отд_ном.* В противном случае для выполнения наиболее часто используемых операций поиска данных о конкретном сотруднике понадобится последовательный просмотр в среднем половины записей файла.

Потребности информационных систем (продолжение)

Требуемая структура файла "Сотрудники" представлена на рисунке 1. Но даже в этом случае, чтобы получить численность отдела или общий размер зарплаты, система должна выбирать все записи о сотрудниках указанного отдела и считать соответствующие общие значения.

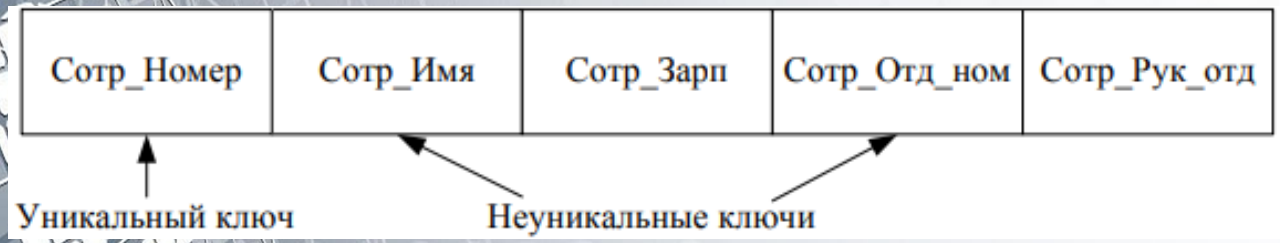


Рисунок 1 – Структура файла «Сотрудники»

Таким образом, при реализации достаточно простой ИС на базе файловой системы возникают следующие затруднения:

– требуется создание достаточно сложной надстройки для многоключевого доступа к файлам;

– возникает существенная избыточность данных (для каждого сотрудника повторяется имя руководителя его отдела);

– требуется выполнение массовой выборки и вычислений для получения суммарной информации об отделах.

Потребности информационных систем (продолжение)

Для улучшения ситуации можно поддерживать два файла "Сотрудники" и "Отделы". Файл "Сотрудники" должен содержать поля – номер удостоверения (*Сотр_Номер*), полное имя (*Сотр_Имя*), размер зарплаты (*Сотр_Зарп*), номер отдела (*Сотр_Отд_ном*), а файл "Отделы" – номер отдела (*Отд_Номер*) и номер удостоверения сотрудника, являющегося руководителем отдела (*Отд_Рук*). Дополнительно файл "Отделы" может содержать поля – общий размер зарплаты сотрудников отдела (*Отд_Общ_зарп*) и общее количество сотрудников в отделе (*Отд_Штат*). Структура этих файлов представлена на рисунке 2.

Введение двух файлов позволит преодолеть большинство затруднений: во-первых, каждый из файлов будет содержать недублируемую информацию, и, во-вторых, не возникнет необходимости в динамических вычислениях суммарной информации по отделам.

Рисунок 2 – Структура файлов "Сотрудники" и "Отделы"

Потребности информационных систем (продолжение)

Следует отметить, что при таком подходе ИС должна обладать некоторыми новыми особенностями, сближающими ее с СУБД. Система должна знать, что она работает с двумя информационно-связанными файлами (это шаг в сторону схемы БД), должна иметь информацию о структуре и смысле каждого поля. Например, системе должно быть известно, что у полей *Сотр_Отд_ном* в файле "Сотрудники" и *Отд_Номер* в файле "Отделы" один и тот же смысл. Также система должна учитывать, что в ряде случаев изменение данных в одном файле должно автоматически вызывать модификацию второго файла, чтобы общее содержимое файлов было согласованным. Например, если на работу принимается новый сотрудник, то нужно добавить запись в файл "Сотрудники", а также изменить поля *Отд_Общ_зарп* и *Отд_Штат* в записи файла "Отделы", соответствующей отделу данного сотрудника. Понятие согласованности (или целостности) данных является ключевым понятием БД. Фактически, если ИС поддерживает согласованное хранение данных в нескольких файлах, можно говорить о том, что она поддерживает БД. Если же некоторая система управления данными позволяет работать с несколькими файлами, обеспечивая их согласованность, то ее можно назвать СУБД. Требование поддержания согласованности данных в нескольких файлах не позволяет при построении ИС обойтись библиотекой функций, поскольку такая система должна обладать некоторыми собственными данными (метаданными), определяющими целостность данных.

Потребности информационных систем (окончание)

Другой пример – в ИС требуется обеспечить параллельную работу с БД. Если опираться на использование файлов, то для обеспечения корректности на все время модификации любого из двух файлов доступ других пользователей к этому файлу будет блокирован. Таким образом, зачисление на работу нового сотрудника существенно затормозит получение информации о другом сотруднике, даже если они работают в разных отделах.

Для примера представим, что в ИС обрабатывается операция принятия на работу нового сотрудника. Следуя требованиям согласованного изменения файлов, ИС вставляет новую запись в файл "Сотрудники" и собирается модифицировать соответствующую запись файла "Отделы", но именно в этот момент происходит аварийное выключение питания ЭВМ.

Очевидно, что после перезапуска системы ее БД будет находиться в несогласованном состоянии. Потребуется выяснить это и привести данные в согласованное состояние. СУБД берет такую работу на себя, поддерживая транзакционное управление и журнализацию изменений БД. Прикладная система не обязана заботиться о поддержке корректности состояния БД, хотя и должна знать, какие цепочки операций изменения данных являются допустимыми

СУБД как независимый системный компонент

Появление СУБД в качестве отдельного компонента было вызвано необходимостью разрешить проблемы, характерные для файловых систем, большинство из которых являются следствием того, что определение данных (метаданные) содержится внутри приложений, а не хранится отдельно и независимо от них (рис. 3, а). СУБД в качестве отдельного компонента выполняют функции по взаимодействию между БД с одной стороны и прикладными ИС с другой стороны, выступая в роли сервера, обсуживающего нескольких клиентов – прикладных ИС (рис. 3, б).

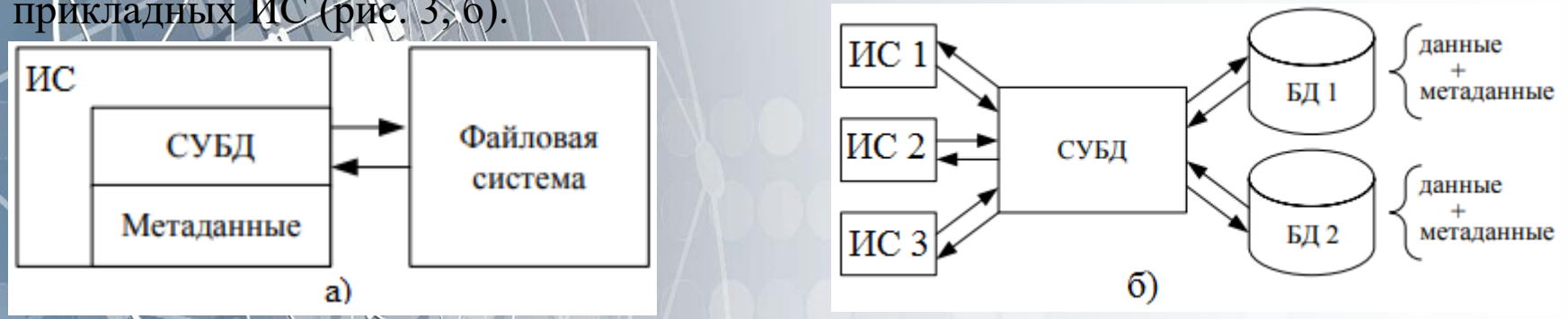


Рисунок 3 – СУБД в составе ИС (а) и в качестве отдельного компонента (б)

СУБД представляет собой программное обеспечение, с помощью которого можно определять, создавать и поддерживать БД, а также осуществлять к ним контролируемый доступ.

СУБД как независимый системный компонент (окончание)

Основные возможности СУБД, которые не обеспечивают файловые системы:

7. Поддержка резервного копирования и восстановления. Ответственность за обеспечение защиты данных от сбоев аппаратного и программного обеспечения в файловых системах возлагается на пользователя. Может потребоваться периодически выполнять резервное копирование данных. При этом в случае сбоя может быть восстановлена резервная копия, но результаты работы, выполненной после резервного копирования, будут утрачены, и данную работу потребуется выполнить заново. В современных СУБД предусмотрены средства снижения вероятности потерь информации при возникновении различных сбоев.

8. Обеспечение повышенной безопасности данных путем предоставления имен и паролей для идентификации зарегистрированных пользователей, а также разграничения доступа к данным со стороны зарегистрированных пользователей.

9. Улучшение производительности. В СУБД предусмотрено много стандартных функций, которые программист должен самостоятельно реализовать в приложениях для файловых систем. На базовом уровне СУБД обеспечивает все низкоуровневые процедуры работы с файлами, которую обычно выполняют приложения. Наличие этих процедур позволяет программисту сконцентрироваться на разработке более специальных, необходимых пользователям функций, не заботясь о подробностях их воплощения на более низком уровне.

2. Архитектура системы баз данных

Трехуровневая архитектура ANSI-SPARC

Основная цель СУБД заключается в том, чтобы предложить пользователю абстрактное представление данных, скрыв конкретные особенности хранения и управления ими. Поскольку БД является общим ресурсом, то каждому пользователю может потребоваться свое, отличное от других представление о характеристиках информации, сохраняемой в БД. Для удовлетворения этих потребностей архитектура большинства современных СУБД в той или иной степени строится на основе архитектуры ANSI-SPARC, предложенной Комитетом планирования стандартов и норм (Standards Planning and Requirements Committee – SPARC) Национального института стандартизации США (American National Standard Institute – ANSI). **Архитектура ANSI/SPARC включает три уровня: внутренний, концептуальный и внешний** (рис. 4).

Внешний уровень – индивидуальный уровень отдельного пользователя (или приложения), где каждый пользователь (приложение) имеет свое видение данных. Этот уровень определяет точку зрения на БД отдельных пользователей. Каждое приложение видит и обрабатывает только те данные, которые необходимы именно этому приложению. Каждый пользователь имеет дело с представлением реального мира, выраженным в наиболее удобной для него форме. Внешнее представление содержит только ту информацию, которая интересует пользователя, хотя в целом БД может содержать и другую информацию, о существовании которой пользователь может и не подозревать.

Архитектура системы баз данных (продолжение)

Помимо этого, различные представления могут по-разному отображать одни и те же данные. Например, один пользователь может просматривать даты в формате «день, месяц, год», а другой – в формате «год, месяц, день». Некоторые представления могут включать производные или вычисляемые данные, которые не хранятся в БД как таковые, а создаются по мере необходимости.

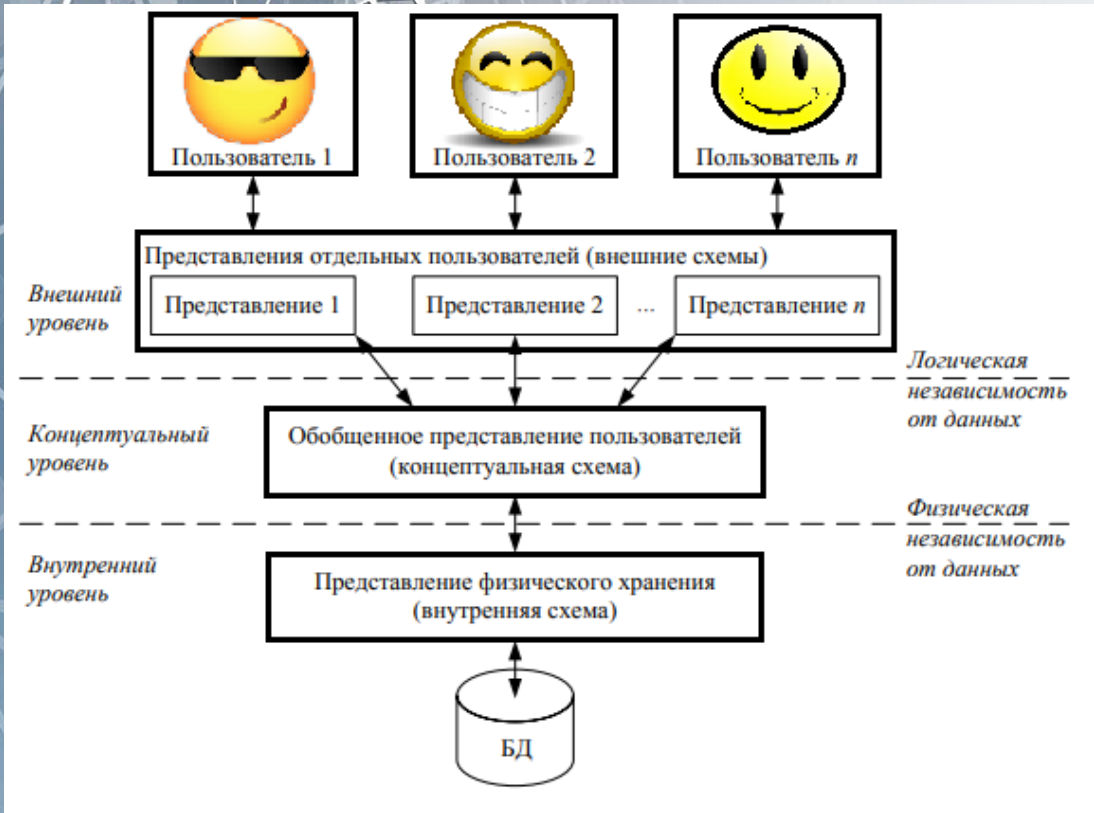


Рисунок 4 – Трехуровневая архитектура ANSI-SPARC

Концептуальный уровень является промежуточным уровнем в трехуровневой архитектуре и содержит логическую структуру БД. На концептуальном уровне представлены все сущности, их атрибуты и связи, накладываемые на данные ограничения, а также информация о мерах обеспечения безопасности и поддержки целостности данных.

Архитектура системы баз данных (окончание)

Внутренний уровень – это низкоуровневое представление БД, наиболее близкое к физическому хранилищу информации, т.е. связан со способами сохранения информации на физических устройствах (внешних носителях информации).

Архитектура ANSI-SPARC позволяет обеспечить независимость от данных, которая означает, что изменения на нижних уровнях не влияют на верхние уровни. Различают **два типа независимости от данных**: **логическую** и **физическую**.

Логическая независимость от данных означает полную защищенность внешних представлений пользователей от изменений, вносимых в концептуальную схему. Добавление или удаление новых сущностей, атрибутов или связей, должны осуществляться без необходимости корректировки прикладных программ. Очевидно, что пользователи, для которых эти изменения предназначались, должны знать о них, но очень важно, чтобы другие пользователи даже не подозревали об этом.

Физическая независимость от данных означает защищенность концептуальной схемы от изменений, вносимых во внутреннюю схему. Физическая независимость предполагает возможность переноса хранимой информации с одних носителей на другие при сохранении работоспособности всех приложений, работающих с БД. Такие изменения внутренней схемы, как использование различных файловых систем, различных устройств хранения, модификация индексов или хеширование, должны осуществляться без необходимости внесения изменений в концептуальную схему. Пользователем могут быть замечены изменения только в общей производительности системы.

Обобщенная архитектура системы баз данных

Процесс функционирования СУБД при обработке простейшего запроса на получение данных предусматривает выполнение следующих шагов:

1. Пользователь посылает СУБД SQL-запрос на получение данных из БД.
2. На основе метаданных СУБД производит анализ прав доступа пользователя и подтверждает или запрещает доступ данного пользователя к запрашиваемым данным. Метаданные содержат информацию об используемых структурах данных, логической организации данных, правах доступа пользователей и физическом расположении данных.
3. В случае запрета на доступ к данным СУБД сообщает пользователю об этом и прекращает дальнейший процесс обработки запроса. В противном случае СУБД производит поиск эффективного способа выполнения запроса, определяет местоположение запрашиваемых данных на физическом уровне (файлы или блоки файловой системы) и пересылает их из устройств хранения в оперативную память.
4. СУБД выбирает из информации, находящейся в оперативной памяти, только те данные, которые требуются в соответствии с условиями запроса, и предоставляет эти данные пользователю.

Обобщенная архитектура системы баз данных (продолжение)

Механизм прохождения запроса в реальных СУБД гораздо сложнее. В представленной схеме функционирования СУБД не учтены функции по контролю над восстановлением данных, проверке целостности данных и некоторые другие. Однако даже эта упрощенная схема позволяет получить представление о том, насколько сложными должны быть механизмы обработки запросов в СУБД.

Рассмотрев отдельные аспекты функционирования СУБД, можно выделить ее основные архитектурные особенности:

1. СУБД должна управлять как внешней памятью, так и оперативной памятью.
2. СУБД должна иметь средства, обеспечивающие ее основную функциональность, т.е. средства анализа и оптимизации запросов, контроля целостности данных, протоколирования, восстановления после сбоев и др.

Обобщенная архитектура СУБД представлена на рисунке 5. ➔

Обобщенная архитектура системы баз данных (продолжение)

Условно оперативную память, которой управляет СУБД, можно представить как совокупность буферов, хранящих данные, журналы транзакций, планы выполнения запросов, таблицы блокировок и другие фрагменты системного каталога (словаря данных), которые необходимы для обработки запросов пользователей. Большая часть обращений к СУБД представляет собой инициированные пользователями или приложениями SQL-запросы, которые поступают на обработку в процессор запросов. Процессор запросов представлен двумя компонентами компилятором запросов и исполняющей машиной.

Компилятор запросов выполняет начальную процедуру контроля прав доступа пользователя или приложения, в ходе которой проверят наличие соответствующих полномочий для выполнения затребованных операций с данными. Если пользователю или приложению разрешено выполнение затребованных операций, то компилятор запросов транслирует поступивший SQL-запрос во внутренний формат СУБД – план запроса, который описывает последовательность инструкций, подлежащих выполнению. Инструкции плана запроса представляют собой операции реляционной алгебры.

Обобщенная архитектура системы баз данных (продолжение)

Компилятор запросов состоит из трех основных частей:

– синтаксического анализатора, выполняющего синтаксический анализ запроса и создающего на основе текста запроса древовидную структуру данных;

– семантического анализатора, выполняющего семантический анализ запроса (проверку того, все и отношения и их атрибуты, упомянутые в тексте запроса, действительно существуют в БД) и функции преобразования дерева, построенного синтаксическим анализатором, в дерево алгебраических операторов, отвечающих исходному плану запроса;

– оптимизатора запросов, осуществляющего трансформацию плана запроса, в котором не определен жесткий порядок выполнения элементарных операций над исходными объектами, в наиболее эффективную последовательность фактических операций над данными.

Оптимизатор запросов строит все возможные планы выполнения запросов и для каждого из них производит стоимостные оценки по быстродействию. Принимая решение о том, какая из последовательностей операций с большей вероятностью окажется самой оптимальной по быстродействию, оптимизатор запросов пользуется метаданными и накопленной статистической информацией. Например, наличие индекса способно существенным образом повлиять на выбор наиболее эффективного плана.

Обобщенная архитектура системы баз данных (продолжение)

Если пользователь выполняет запрос, план выполнения которого уже хранится в буфере оперативной памяти, то компилятор запросов не производит его разбор и построение нового плана, он сразу запускает его на выполнение, возможно, с новыми параметрами.

Исполняющая машина несет ответственность за осуществление всех операций, предусмотренных планом запроса. В процессе своей работы она взаимодействует с большинством компонентов СУБД. Чтобы получить возможность обрабатывать данные, исполняющая машина обязана считать их с внешнего носителя информации и перенести в буферы оперативной памяти. При этом ей необходимо взаимодействовать с модулем управления транзакциями, чтобы избежать опасности обращения к блокированным порциям информации или получить гарантию того, что все изменения, внесенные в БД, зафиксированы в протоколе.

Модуль управления транзакциями поддерживает механизмы фиксации и отката транзакций и обеспечивает сохранение всей информации, которая требуется для восстановления системы после сбоев. Кроме того, модуль управления транзакциями поддерживает специальные механизмы поиска тупиковых ситуаций или взаимоблокировок и реализует одну из принятых стратегий принудительного завершения транзакций для развязывания тупиковых ситуаций. Модуль управления транзакциями представлен в виде двух основных компонентов – планировщика заданий и модуля протоколирования и восстановления.

Обобщенная архитектура системы баз данных (окончание)

Модуль протоколирования и восстановления предназначен для исключения негативных последствий, вызванных системными сбоями во время выполнения транзакций, и обеспечивает выполнение требования устойчивости транзакций. С целью удовлетворения данного требования блок протоколирования фиксирует каждое изменение БД в специальных (журнальных) файлах, а блок восстановления в случае возникновения нештатных ситуаций способен считать журнальный файл, содержащий протокол изменений БД, и привести БД в согласованное состояние. Протокол изменений БД изначально сохраняется в буферах оперативной памяти, а затем блок протоколирования в определенные моменты времени взаимодействует с модулем управления данными во внешней памяти с целью сохранения содержимого буферов во внешней памяти.

Поскольку транзакции состязаются за ресурсы, которые могут быть блокированы планировщиком заданий, возможно возникновение таких ситуаций, когда ни одна из транзакций не в состоянии продолжить работу ввиду того, что ей необходим ресурс, находящийся в ведении другой транзакции. Планировщик заданий обладает прерогативой вмешиваться в ситуацию и прерывать (откатывать) одну или несколько транзакций, чтобы позволить остальным продолжить работу.

Логические структуры базы данных

Таблица – это базовая структура данных в любой реляционной СУБД. Таблица представляет собой набор строк. Каждая таблица состоит из одного или нескольких столбцов, каждому из которых назначается имя и тип данных. Тип данных столбца определяет точность представления информации, которая сохраняется в этом столбце.

Представление – структура данных, определяемая SQL-командой, которая хранится в БД. Сами представления не содержат данных, а лишь позволяют взглянуть на них так, как это определено в запросе. Представление строится над набором базовых таблиц, каждая из которых может быть либо реальной таблицей базы данных, либо другим представлением.

Представления используются в целях:

– упрощения доступа к данным;

– реализации специальных требований к безопасности данных;

– скрытия от приложения точной структуры базовых таблиц.

Материализованные представления по существу не являются представлениями в описанном выше смысле, а являются физическими таблицами, в которых хранятся заранее агрегированные данные, что позволяет заметно повысить производительность хранилищ данных.

Логические структуры базы данных (продолжение)

Индекс – это структура данных, ускоряющая доступ к строкам таблицы. Индекс ассоциируется с одной таблицей и содержит данные из одного или нескольких столбцов. Поскольку в индексе хранится меньше данных, чем в полной строке таблицы, и поскольку индексы организованы в виде специальной структуры, ускоряющей чтение, то для доступа к данным по значениям индексированных столбцов требуется меньше операций ввода/вывода и выборка строк может оказаться быстрее.

Индекс может быть уникальным (т.е. в таблице не может быть двух строк с одинаковым значением индексного ключа) или неуникальным. Строки, для которых значение индексного ключа равно NULL, не включаются в индекс.

Способы организации индексов:

– стандартные B*-деревья;

– индексы по реверсированному ключу;

– битовые индексы.

В реляционных СУБД наиболее часто индекс имеет структуру B*-дерева. B*-дерево состоит из одного или нескольких промежуточных уровней и одного листового уровня. Промежуточные узлы содержат информацию о диапазоне значений, находящихся на следующем промежуточном уровне.

Логические структуры базы данных (продолжение)

Количество промежуточных уровней между корнем и листовым уровнем называется глубиной индекса. На листовом уровне находятся узлы, содержащие сами индексированные значения и ROWID ассоциированных с ними строк таблицы (рис. 6).

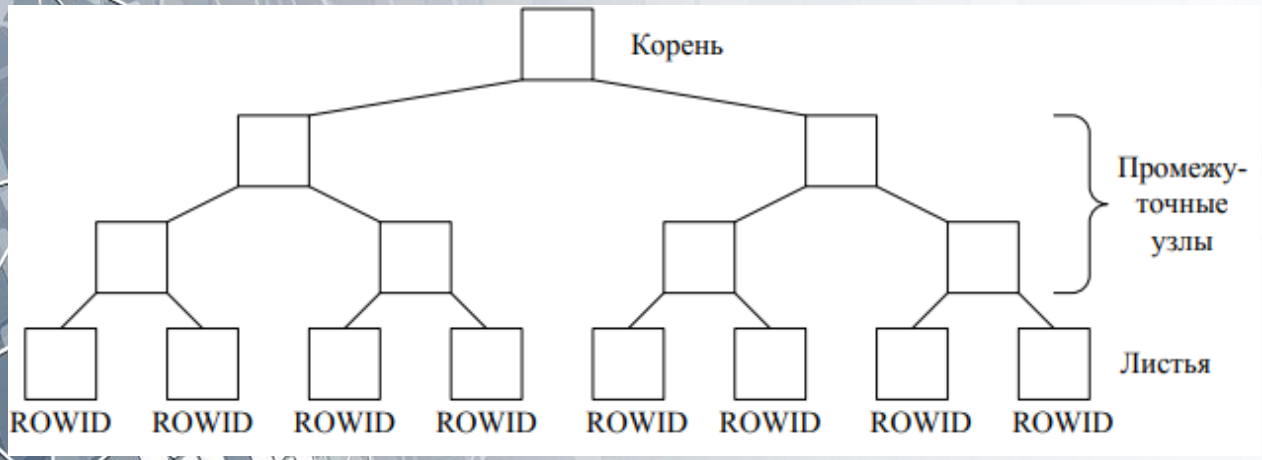


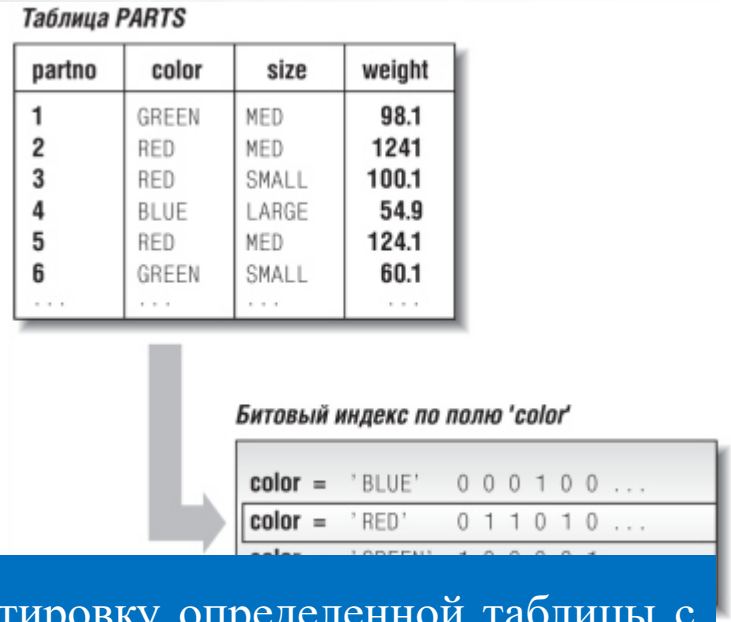
Рисунок 6 – Индекс на базе B*-дерева

На верхних уровнях B*-дерева узлов мало, поэтому для спуска по дереву требуется относительно небольшое число операций ввода/вывода. Все листовые узлы расположены в индексе на одной и той же глубине, поэтому для получения любой индексной записи требуется одно и то же количество операций. Это выравнивает производительность доступа по индексу.

В индексе по реверсированному ключу порядок байтов в хранимом ключе меняется на обратный, что позволяет исправить дисбаланс, вызванный добавлением монотонно возрастающих значений (последовательные порядковые номера или увеличивающиеся даты всегда добавляются с правой стороны индекса) в стандартное B*-дерево.

Логические структуры базы данных (продолжение)

В стандартном B*-дереве значения ROWID хранятся в листовых узлах индекса. В битовом индексе каждый бит представляет собой один ROWID (рис. 7). Если строка содержит определенное значение, соответствующий этой строке бит «поднят». Для преобразования номера бита в ROWID применяется функция отображения. В отличие от других типов, в битовых индексах представлены и строки, содержащие NULL в качестве значения ключа. Для хранения битового индекса с небольшим числом значений требуется гораздо меньше места, чем для стандартного B*-дерева. Возможности битовых индексов особенно важны при организации



2. **Некластеризованный индекс** осуществляет сортировку определенной таблицы с помощью дерева индексов. При этом строки таблицы на диске могут находиться в любом порядке. Пример: предметный указатель в конце книги.

1. **Кластеризованный индекс** физически сортирует строки таблицы на диске в соответствии с индексируемым полем. Кластеризованный индекс связывает листовые узлы страниц индексов со страницами данных, обеспечивая тот же порядок данных, что и в индексе. Таблицы БД могут иметь только один физический порядок сортировки и, следовательно, только один кластеризованный индекс. Пример: телефонный справочник.

Логические структуры базы данных (продолжение)

В кластеризованном индексе на физической странице листового узла расположены непосредственно строки данных, а в некластеризованном индексе – только ссылка на страницу со строками данных (рис. 8).

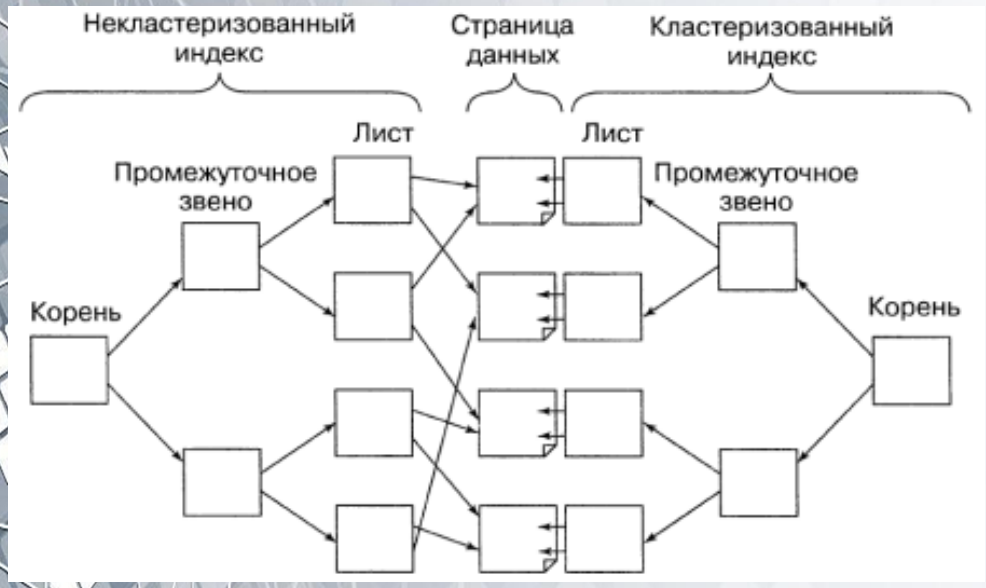


Рисунок 8 – Принципы организации кластеризованного и некластеризованного индексов

Последовательности предназначены для порождения уникальных числовых значений для ключей или идентификаторов. Последовательности существуют независимо от таблиц, так что одну последовательность можно использовать для нескольких таблиц. Синонимы предназначены для упрощения имен, сделав их более удобными для восприятия. Синоним может быть публичным (тогда к нему может обратиться любой пользователь БД) или приватным (доступным только владельцу содержащей его схемы).

Логические структуры базы данных (продолжение)

Хранимая процедура – хранимая в БД программа, содержащая набор операторов языка SQL и команд, управляющих ходом выполнения программы. Хранимые процедуры хранятся в БД в скомпилированном виде и играют ключевую роль в повышении быстродействия. В процедуре предоставляется возможность:

– использовать операторы, которые выполняют любые операции в БД (выборка, вставка, изменение или удаление данных), включая возможность вызова других процедур;

– принимать заранее определенные переменные во входные параметры и использовать их в своих расчетах;

– возвращать значения в форме выходных параметров.

Функция – хранимая в БД программа, содержащая набор операторов языка SQL и команд, управляющих ходом выполнения программы, возвращающая определенное значение. Функции могут получать некоторое количество параметров (возможно, ни одного) и возвращать скалярное значение или таблицу.

Триггер – специальный вид хранимой процедуры, выполняемый всякий раз, когда с таблицей БД происходит некоторое событие. Срабатывание триггера могут вызвать события обновления, вставки или удаления данных из БД. Моменты срабатывания триггеров – перед, после и вместо выполнения события, вызывающего его срабатывание. Триггеры представляют собой удобное средство для обеспечения целостности данных.

Логические структуры базы данных (окончание)

Кластер – структура данных, повышающая производительность выборки. С помощью кластеров можно хранить взаимосвязанные данные в смежных областях диска. СУБД считывает данные поблочно, поэтому хранение значений рядом сокращает количество операций ввода/вывода, необходимых для их выборки, так как в одном блоке могут содержаться взаимосвязанные строки.

Кластер состоит из одной или нескольких таблиц. В состав кластера входит кластерный индекс, в котором хранятся все значения соответствующего кластерного ключа. Каждое значение в кластерном индексе указывает на блок данных, содержащий только строки с тем же значением кластерного ключа. Если кластер содержит несколько таблиц, то таблицы должны быть соединены, а кластерный индекс должен содержать значения столбцов, по которым производится соединение.

Хешированный кластер обладает существенной особенностью, которая делает его еще быстрее. Каждый запрос данных из кластеризованной таблицы требует по меньшей мере двух операций ввода/вывода – для кластерного индекса и для данных. В хешированном кластере связанные строки данных хранятся вместе, но группируются согласно хеш-значению кластерного ключа. Это значение вычисляется хеш-функцией, т.е. каждая операция выборки начинается с вычисления хеш-значения, а затем сразу же выбирается блок данных, содержащий нужные строки.

Заключение

СУБД представляет собой эффективный инструмент сбора больших объемов информации, позволяющий сохранять данные в целостности и безопасности на протяжении длительного времени. СУБД предлагает пользователям следующие функциональные возможности:

1. Средства постоянного хранения данных. СУБД, подобно файловым системам поддерживают возможности хранения больших фрагментов данных, которые существуют независимо от процессов их использования. СУБД значительно превосходят файловые системы в отношении гибкости представления информации, предлагая структуры, обеспечивающие эффективный доступ к большим объемам данных.
2. Интерфейс программирования. СУБД позволяет пользователю или прикладной программе обращаться к данным и изменять их посредством команд развитого языка запросов. Преимущества СУБД в сравнении с файловыми системами проявляются и в том, что первые дают возможность манипулировать данными самыми разнообразными способами, гораздо более гибкими, чем обычные операции чтения и записи файлов.
3. Управление транзакциями. СУБД поддерживают параллельный доступ к данным, т.е. возможность единовременного обращения к одной и той же порции данных со стороны нескольких различных процессов, называемых транзакциями.

Спасибо за внимание!