Содержание

1 Экзамен

1.1 Базы данных и системы управления базами данных. Определения, основные функции и классификация

База данных (БД) — самодокументированное собрание интегрированных записей.

1) БД является самодокументированной, т.е. она содержит описание собственной структуры. Это описание называется словарем данных, каталогом данных или метаданными.
2) БД является собранием интегрированных записей, т.е. она содержит:
   - файлы данных;
   - метаданные (данные о данных);
   - индексы, которые представляют связи между данными, а также служат для повышения производительности приложений базы данных;
   - метаданные приложения.
3) БД является информационной моделью пользовательской модели предметной области.

OLAP — online analtyc proccesing (операциооные данные) чтение, вставка, удаление, мин время отклика.

OLTP — online transaction proccessing (перманентые данные) вставка, удаление, обнровление, гарантия изменения информации.

Таблица 1

OLAP	OLTP
чтение	вставка, удаление, обновление
минимальное время отклика	минимальное время вставки, удаление, обновление

Транзакции — либо все действия, либо никакие действия.

Любая бд хранит:

1) метаданные (данные о данных);
2) файлы данных,
3) Индексы (indexes), которые представляют связи между данными, а также служат для повышения производительности приложений базы данных.
4) Может содержать метаданные приложений (application metadata).

Основные храктеристики, требования

1) **Неизбыточность данных** — каждое данное присутствует в БД в единственном экземпляре.

2) **Совместное использование данных** многими пользователями.
3) **Эффективность доступа** к БД - высокое быстродействие, т. е. малое время отклика на запрос.
4) **Целостность данных** — соответствие имеющейся в БД информации её внутренней логике, структуре и всем явно заданным правилам.
5) **Безопасность данных** — защита данных от преднамеренного или непреднамеренного искажения или разрушения данных.
6) **Восстановление данных** после программных и аппаратных сбоев.
7) **Независимость данных** от прикладных программ.

Система баз данных (СБД) — совокупность одной или нескольких баз данных и комплекса информационных, программных и технических средств, обеспечивающих накопление, обновление, корректировку и многоаспектное использование данных в интересах пользователей.

Система управления базами данных (СУБД) — приложение, обеспечивающее создание, хранение, обновление и поиск информации в базах данных.

1.1.1 Основные функции СУБД

1) Управление данными во внешней памяти.
2) Управление буферами оперативной памяти.
3) Управление транзакциями.
4) Журнализация.
5) Поддержка языка или языкового пакета (-ов).

1.1.2 Классификация СУБД

1) По модели данных:
   - Дореляционные (Инвертированные списки, иерархические и сетевые)
     - Инвертированные списки (файлы). БД на основе инвертированных списков представляет собой совокупность файлов, содержащих записи (таблиц). Для записей в файле определен некоторый порядок, диктуемый физической организацией данных. Для каждого файла может быть определено произвольное число других упорядочений на основании значений некоторых полей записей (инвертированных списков). Обычно для этого используются индексы. В такой модели данных отсутствуют ограничения целостности как таковые. Все ограничения на возможные экземпляры БД задаются теми программами, которые работают с БД. Одно из немногих ограничений, которое все-таки может присутствовать - это ограничение, задаваемое уникальным индексом.

  - Иерархичекие
  - Сетевые (могут быть представлены в виде графа; логика выборки зависит от физической организации данных)
- Реляционные
  - Структурный (данные — набор отношений)
  - Целостностный (отношения (таблицы) отвечают определенным условиям целостности)
  - Манипуляционный (манипулирования отношениями осуществляется средствами реляционной алгебры и/или реляционного исчисления)
- Постреляционные

2) По архитектуре организации хранения данных:
   - Локальные (все части локальной СУБД размещаются на одном компьютере)
   - Распределенные (части СУБД могут размещаться на 2-х и более компьютерах)
3) По способу доступа к БД:
   - Файл-серверные (при работе с базой, данные перегоняются приложению, которое с ней работает, вне зависимости от того, сколько их нужно. Все операции — на стороне клиента. Файловый сервер периодически обновляется тем же клиентом)
   - Клиент-серверные (вся работа на сервере, по сети передаются результаты запросов, гораздо меньше информации. Обеспечивается безопасность данных, потому что все происходит на стороне сервера. Проще исключить одновременное изменение и тп)
   - Встраиваемые — библиотека, которая позволяет унифицированным образом хранить большие объемы данных на локальной машине. Доступ к данным может происходить через SQL либо через особые функции СУБД. Встраиваемые СУБД быстрее обычных клиент-серверных и не требуют установки сервера, поэтому востребованы в локальном ПО, которое имеет дело с большими объемами данных.
   - Сервисно-ориентированные (БД является хранилищем сообщений, промежуточных состояний, метаинформации об очередях сообщений и сервисах)
   - Прочие (пространственная, временная и пространственно-временная)

1.1.3 Архитектура хранения данных

1) Локальные.
2) Распределенные.
3) По способу обращения к данным.
   - Файл серверные.
   - Клиент серверные (PostGress, MSSQL, Oracle, MySQL, Mongo).

- Встраиваемые (SQLlite).
- Сервисно-ориентированные (KafcaBD).
- Прочее - time series.

1.2 Семантическое моделирование данных

Любая развитая семантическая модель данных, как и реляционная модель, включает структурную, манипуляционную и целостную части. Главным назначением семантических моделей является обеспечение возможности выражения семантики данных. На практике семантическое моделирование используется на первой стадии проектирования базы данных. При этом в терминах семантической модели производится концептуальная схема базы данных, которая затем:

1) Либо вручную преобразуется к реляционной схеме;
2) Либо реализуется автоматизированная компиляция концептуальной схемы в реляционную;
3) Либо происходит работа с базой данных в семантической модели, т.е. под управлением СУБД, основанных на семантических моделях данных.

Наиболее известным представителем класса семантических моделей предметной области является модель «сущность-связь» или ER-модель, , предложенная Питером Ченом в 1976 году.

Модель сущность-связь — модель данных, позволяющая описывать концептуальные схемы предметной области. Предметная область — часть реального мира, рассматриваемая в пределах данного контекста. Под контекстом здесь может пониматься, например, область исследования или область, которая является объектом некоторой деятельности. ER-модель используется при высокоуровневом (концептуальном) проектировании баз данных.

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

Сущность - это реальный или представляемый объект, информация о котором должна сохраняться и быть доступна. При этом имя сущности - имя типа, а не некоторого конкретного экземпляра этого типа. Каждый экземпляр сущности должен быть отличим от любого другого экземпляра этой сущности.

Связь - это ассоциация, устанавливаемая между сущностями. Эта ассоциация может существовать между разными сущностями или между сущностью и ей же самой (рекурсивная связь). Сущности, включенные в связь, называются её участниками, а количество участников - степенью связи. Связи в ER-модели могут иметь тип «один к одному», «один ко многим», «многие ко многим».

Свойством/атрибут сущности (и связи) является любая деталь, которая служит для уточнения, идентификации, классификации, числовой характеристики или выражения состояния

сущности (или связи).

Ключи: Первичный ключ — набор атрибутов однозначно идентифицирующий кортеж значений, и Внешний ключ

1.3 Реляционная модель данных: структурная, целостная, манипуляционная части. Реляционная алгебра. Исчисление кортежей

Реляционная модель данных согласно трактовке Кристофера Дейта состоит из трех частей: структурной, целостной и манипуляционной.

Структурная часть описывает из каких объектов состоит реляционная модель. Основной структурой данных в реляционной модели является нормализованные n-мерные отношения и основными понятиями структурной части реляционной модели является:

- *Тип данных* — множество значений и операций над ними. (понятие такое же как и в языках программирования);
- *Домен* можно считать уточнением типа данных и рассматривать как подмножество значений некоторого типа данных, имеющий определенный смысл. Характеризуется следующими свойствами:
  - имеет уникальное имя в пределах базы данных;
  - определен на некотором типе данных или на другом домене;
  - может иметь логическое условие, позволяющее описать подмножество данных, доступных для данного домена;
  - несет определенную смысловую нагрузку

  Домен отражает семантику, определенной предметной области и может быть не сколько доменов совпадающих как подмножество, но с различным смыслом. Основное значения домена ограничивается сравнением.
- *Атрибут отношения* — это пара вида <имя_атрибута, имя_домена>, при этом имена атрибутов должны быть уникальны в пределах отношения, но могут совпадать с именем домена.
- *Схема отношения* — это именованное множество упорядоченных пар <имя_атрибута, имя_домена>.
- *Схема БД* — это множество именованных схем отношений.
- *Кортеж* — это множество упорядоченных пар <имя_атрибута, значение_атрибута>, которое содержит одно вхождение каждого имени атрибута, принадлежащего схеме отношения.
- *Отношение* определенное на множестве из n доменов (не обязательно различных), содержит две части: заголовок (схему отношения) и тело (множество из m кортежей). Значения

n и m называются соответственно степенью и кардинальностью отношения.

- Непустое подмножество множества атрибутов схемы отношения будет *потенциальным ключом* тогда и только тогда, когда оно будет обладать свойствами уникальности (в отношении нет двух различных кортежей с одинаковыми значениями потенциального ключа) и неизбыточности (никакое из собственных подмножеств множества потенциального ключа не обладает свойством уникальности).
- В реляционной модели по традиции один из потенциальных ключей должен быть выбран в качестве *первичного ключа*, а все остальные потенциальные ключи будут называться *альтернативными*.
- *Реляционная база данных* — это набор отношений, имена которых совпадают с именами схем отношений в схеме базы данных.

Целостностная часть описывает ограничения специального вида, которые должны выполняться для любых отношений в любых реляционных базах данных. Это целостность сущностей и целостность внешних ключей.

Манипуляционная часть описывает два эквивалентных способа манипулирования реляционными данными - реляционную алгебру и реляционное исчисление.

Реляционная алгебра является основным компонентом реляционной модели, опубликованной Коддом, и состоит из восьми операторов, составляющих две группы по четыре оператора:

- Традиционные операции над множествами: объединение (UNION), пересечение (INTERSECT), разность (MINUS) и декартово произведение (TIMES). Все операции модифицированы, с учетом того, что их операндами являются отношения, а не произвольные множества.
- Специальные реляционные операции: ограничение (WHERE) , проекция (PROJECT), соединение (JOIN) и деление (DIVIDE BY).

Результат выполнения любой операции реляционной алгебры над отношениями также является отношением. Эта особенность называется свойством реляционной замкнутости. Утверждается, что поскольку реляционная алгебра является замкнутой, то в реляционных выражениях можно использовать вложенные выражения сколь угодно сложной структуры.

Исчисление существует в двух формах: исчисление кортежей и исчисление доменов. Основное различие между ними состоит в том, что переменные исчисления кортежей являются переменными кортежей (они изменяются на отношении, а их значения являются кортежами), в то время как переменные исчисления доменов являются переменными доменов (они изменяются на доменах, а их значения являются скалярами).

Выражение исчисления кортежей содержит заключенный в скобки список целевых эле-

ментов и выражение WHERE, содержащее формулу WFF ("правильно построенную формулу"). Такая формула WFF составляется из кванторов (EXISTS и FORALL), свободных и связанных переменных, литералов, операторов сравнения, логических (булевых) операторов и скобок. Каждая свободная переменная, которая встречается в формуле WFF, должна быть также перечислена в списке целевых элементов.

1.4 Теория проектирования реляционных баз данных: функциональные зависимости, нормальные формы

Теория проектирования реляционных баз данных

При проектировании баз данных решаются две основные проблемы:

- проблема логического проектирования баз данных;
- проблема физического проектирования баз данных.

Классический подход к проектированию реляционных баз данных заключается в том, что сначала предметная область представляется в виде одного или нескольких отношений, а далее осуществляется процесс *нормализации* схем отношений, причем каждая следующая нормальная форма обладает лучшими свойствами, чем предыдущая. Каждой нормальной форме соответствует некоторый определенный набор ограничений, и отношение находится в некоторой нормальной форме, если удовлетворяет свойственному ей набору ограничений. В теории реляционных баз данных обычно выделяется следующая последовательность нормальных форм:

1) *Первая нормальная форма* (1НФ/1NF). Отношение находится в 1НФ, если все его атрибуты являются простыми, все используемые домены должны содержать только скалярные значения, при этом нет повторяющихся кортежей.
2) *Вторая нормальная форма* (2НФ/2NF). Отношение находится во 2НФ, если оно находится в 1НФ и каждый не ключевой атрибут неприводимо зависит от первичного ключа, т.е. в составе потенциального ключа отсутствует подмножество атрибутов, от которых также можно вывести неключевые атрибуты.
3) *Третья нормальная форма* (3НФ/3NF). Отношение находится в 3НФ, когда находится во 2НФ и каждый нетразитивно зависит от первичного ключа.
4) *Нормальная форма Бойса-Кодда* (НФБК). Отношение находится в НФБК, когда каждая нетривиальная и неприводимая слева функциональная зависимость обладает потенциальным ключом в качестве детерминанта.
5) *Четвертая нормальная форм* (4НФ/4NF). Отношение находится в НФБК, если оно находится в НФБК и все нетривиальные многознаяные зависимости фактически являются функциональными зависимостями от ее потенциальных ключей.
6) *Пятая нормальная форма* (5НФ/5NF). Отношение находится в 5НФ тогда и только то-

гда, когда каждая нетривиальная зависимость соединения определяется потенциальным ключом этого отношения.

Основные свойства нормальных форм:

1) каждая следующая нормальная форма в некотором смысле лучше предыдущей;
2) при переходе к следующей нормальной форме свойства предыдущих нормальных свойств сохраняются.

Процесс проектирования реляционной базы данных на основе метода нормализации преследует две основные цели: избежать избыточность хранения данных и устранить аномалии обновления отношений (под этим подразумеваются определенные трудности, появляющиеся при выполнении операций обновления INSERT, DELETE, UPDATE).

Наиболее важные на практике нормальные формы отношений основываются на фундаментальном в теории реляционных баз данных понятии функциональной зависимости.

1.4.1 Функциональная зависимость

Пусть R - это отношение, а X и Y - произвольные подмножества множества атрибутов отношения R. Тогда Y *функционально зависимо* (ФЗ) от X, что в символическом виде записывается как $X \to Y \Leftrightarrow \forall$значение множества X связано в точности с одним значением множества Y. (Левая ФЗ часть — детерминант, правая — зависимая часть)

Очевидным способом сокращения размера множества ФЗ было бы исключение тривиальных зависимостей, т. е. таких, которые не могут не выполняться.

Множество всех ФЗ, которые задаются данным множеством ФЗ S, называется замыканием S и обозначается символом $S+$.

Пусть в перечисленных ниже правилах A, B и C — произвольные подмножества множества атрибутов заданной переменной-отношения R, а символическая запись AB означает $\{A, B\}$. Тогда правила вывода определяются следующим образом:

Теорама (Правила вывода).

1) *Правило рефлективности* $(B \subseteq A) \Rightarrow (A \to B)$
2) *Правило дополнения* $(A \to B) \Rightarrow (AC \to BC)$
3) *Правило транзитивности* $(A \to B)\&\&(B \to C) \Rightarrow (A \to C)$

Каждое и этих правил может быть непосредственно доказано на основе определения ФЗ. Более того, эти правила являются полными в том смысле, что для заданного множества ФЗ S минимальный набор ФЗ, которые подразумевают все зависимости из множества S, может быть выведен из S на основе этих правил. Они также являются исчерпывающими, поскольку никакие дополнительные ФЗ (т.е. ФЗ, которые не подразумеваются ФЗ множества S) с их помощью не могут быть выведены. Иначе говоря, эти правила могут быть использованы для получения

замыкания $S+$.

1.4.2 Дополнительные правила вывода.

1) *Правило самоопределения* $(A \to A)$
2) *Правило декомпозиции* $(A \to B) \Rightarrow (A \to B)\&\&(A \to C)$
3) *Правило объединения* $(A \to B)\&\&(A \to c) \Rightarrow (A \to BC)$
4) *Правило композиции* $(A \to B)\&\&(C \to D) \Rightarrow (AB \to CD)$
5) *Общая теорема объединения* $(A \to B)\&\&(C \to D) \Rightarrow (A(C - B) \to BD)$

Два множества ФЗ S1 и S2 эквивалентны тогда и только тогда, когда они являются покрытиями друг для друга, т. е. $S1+ = S2+$.

Множество ФЗ является неприводимым тогда и только тогда, когда оно обладает всеми перечисленными ниже свойствами:

1) Каждая ФЗ этого множества имеет одноэлементную правую часть.
2) Ни одна ФЗ множества не может быть устранена без изменения замыкания этого множества.
3) Ни один атрибут не может быть устранен из левой части любой ФЗ данного множества без изменения замыкания множества.

1.4.3 Общая схема процедуры нормализации

1) Переменную-отношение в 1НФ следует разбить на такие проекции, которые позволят исключить все функциональные зависимости, не являющиеся неприводимыми. В результате будет получен набор переменных отношений в 2НФ.
2) Полученные переменные-отношения в 2НФ следует разбить на такие проекции, которые позволят исключить все существующие транзитивные функциональные зависимости. В результате будет получен набор переменных-отношений в 3НФ.
3) Полученные переменные-отношения в 3НФ следует разбить на проекции, позволяющие исключить любые оставшиеся функциональные зависимости, в которых детерминанты не являются потенциальными ключами. В результате такого приведения будет получен набор переменных-отношений в НФБК. Замечание. Правила 1-3 могут быть объединены в одно: "Исходную переменную-отношение следует разбить на проекции, позволяющие исключить все функциональные зависимости, в которых детерминанты не являются потенциальными ключами".
4) Полученные переменные-отношения в НФБК следует разбить на проекции, позволяющие исключить любые многозначные зависимости, которые не являются функциональными. В результате будет получен набор переменных- отношений в 4НФ.
5) Полученные переменные-отношения в 4НФ следует разбить на проекции, позволяющие исключить любые зависимости соединения, которые не подразумеваются потенциальными ключами. В результате будет получен набор переменных-отношений в 5НФ.

1.5 Теория проектирования хранилищ данных. Основные принципы построения. ETL и ELT процессы

Хранилище данных – это система, в которой собраны данные из различных источников внутри компании и эти данные используются для поддержки принятия управленческих решений.

Трехуровневая архитектура состоит из:

- Нижний уровень содержит сервер базы данных используемый для извлечения данных из множества различных источников.
- Средний уровень содержит сервер OLAP, который преобразует данные в структуру, подходящую для анализа и сложных запросов.
- Верхний уровень — уровень клиента, содержащий инструменты, используемые для высокоуровневого анализа данных, создания отчетов и анализа данных.

Есть два хранилища данных:

1) *Подход Ральфа Кимбалла* основывается на важности витрин данных, которые являются хранилиавми данных, облегчающие отчетность и анализ. При этом организация хранилища пространственная, где существуют «виртуальные» объекты. Коллекция витрин данных, которые могут быть пространственно разобщенны.
2) *Подход Билла Инмона* основывается на том, что хранилище данных является централизованным хранилищем всех корпоративных данных. При таком подходе сначала создают нормализованную модель хранилища данных, шде объектами являются физически целостными. Затем создаются витрины размерных на основе модели хранилища. Это известно как нисходящий подход к хранилищу данных.

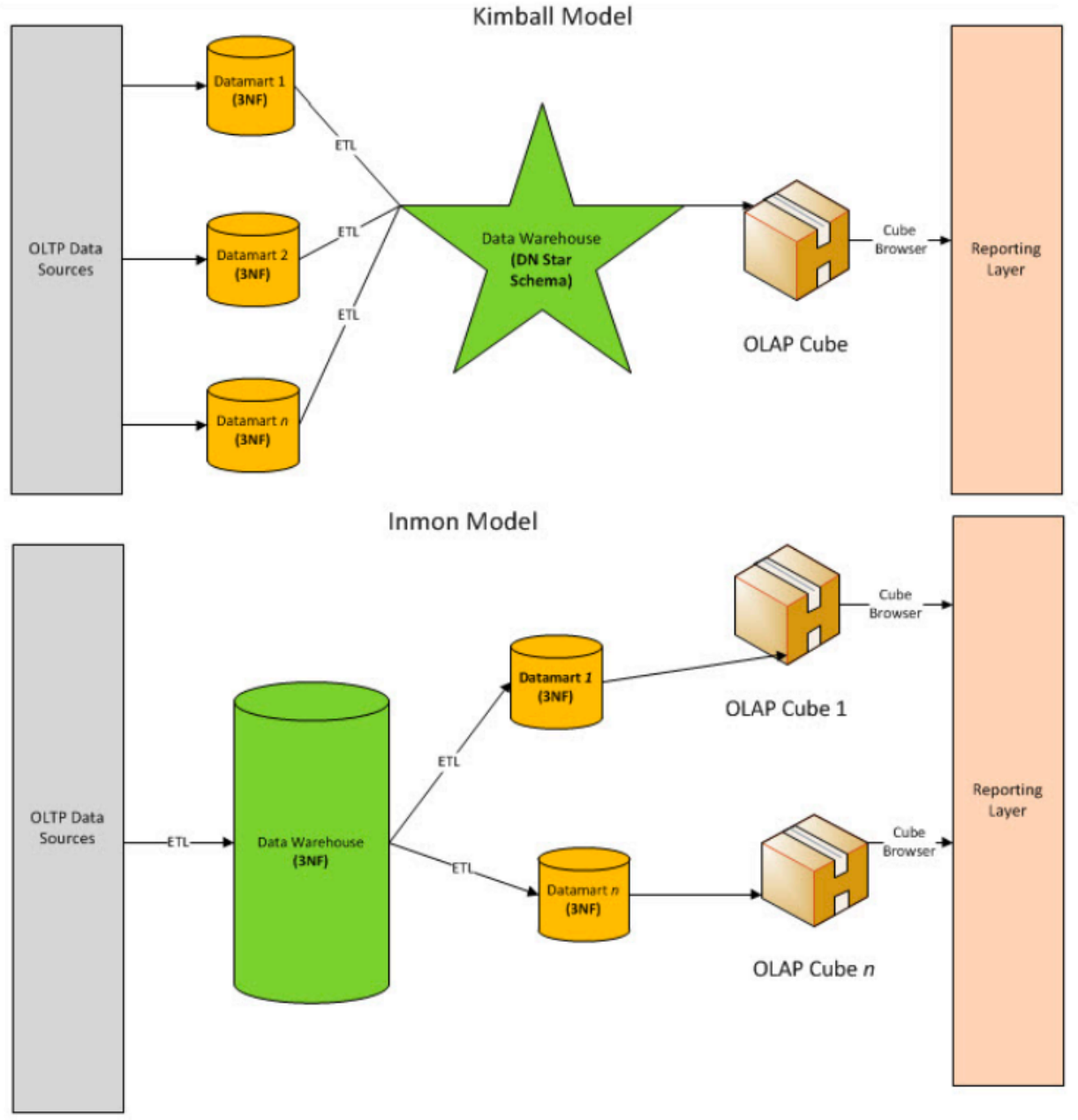

Рисунок 1 – Структуры хранилищ данных Киммбалла/Инмона

Схемы «звезда» и «снежинка» — это два способа структурировать хранилище данных.

Схема типа «звезда» имеет централизованное хранилище данных, которое хранится в таблице фактов. Схема разбивает таблицу фактов на ряд денормализованных таблиц измерений. Таблица фактов содержит агрегированные данные, которые будут использоваться для составления отчетов, а таблица измерений описывает хранимые данные. Денормализованные проекты менее сложны, потому что данные сгруппированы. Таблица фактов использует только одну ссылку для присоединения к каждой таблице измерений. Более простая конструкция звездообразной схемы значительно упрощает написание сложных запросов.

Схема типа «снежинка» отличается тем, что использует нормализованные данные. Нормализация означает эффективную организацию данных так, чтобы все зависимости данных были определены, и каждая таблица содержала минимум избыточности. Таким образом, отдельные таблицы измерений разветвляются на отдельные таблицы измерений.

ETL & ELT

ETL и ELT — два разных способа загрузки данных в хранилище.

ETL (Extract, Transform, Load) начала извлекают данные из пула источников данных. Данные хранятся во временной промежуточной базе данных. Затем выполняются операции преобразования, чтобы структурировать и преобразовать данные в подходящую форму для целевой системы хранилища данных. Затем структурированные данные загружаются в хранилище и готовы к анализу.

Инструменты ETL используются для интеграции данных, чтобы удовлетворить требованиям систем управления реляционными базами данных и/или традиционных хранилищ данных с поддержкой OLAP (online analytical processing, аналитической онлайн-обработки). Инструменты OLAP и запросы (SQL) требуют, чтобы массивы данных структурировались и стандартизировались при помощи серии преобразований, выполняемых до того, как данные попадут в хранилище.

ELT (Extract, Load, Transform) анные сразу же загружаются после извлечения из исходных пулов данных. Промежуточная база данных отсутствует, что означает, что данные немедленно загружаются в единый централизованный репозиторий. Данные преобразуются в системе хранилища данных для использования с инструментами бизнес-аналитики и аналитики.

Этапы ETL & ELT:

1) Процесс загрузки данных из пулла источников.
2) Процесс валидации (отвечает за выявление ошибок и пробелов в данных).
3) Процесс мэппинга данных.
4) Процесс агрегации данных. Этот процесс нужен из-за разности детализации данных в

OLTP и OLAP системах. OLTP система может содержать несколько сумм для одного и того же набора элементов справочников, а OLAP-системы — это, по сути, полностью денормализованная таблица фактов и окружающие ее таблицы справочников.

5) Процесс выгрузки данных в целевую систему.

1.6 Транзакции. Определение, свойства и уровни изоляции транзакций. Неблагоприятные эффекты, вызванные параллельным выполнением транзакци , и способы их устранения. Управление транзакциями и способы обработки ошибок

Транзакция — последовательность операций, выполняемая как единое целое. (всё или ничего).

Для поддержания целостности транзакция должна обладать четырьмя свойствами АСИД:

- *Атомарность.* Транзакция либо выполняется полностью либо не выполняется вовсе.
- *Согласованность (Непротиворечивость).* . При завершении транзакции не должны быть нарушены ограничения накладываемые на данные (например constraints в БД). Согласованность подразумевает, что система будет переведена из одного корректного состояния в другое корректное
- *Изоляция.* Параллельно выполняемые транзакции не должны влиять друг на друга, например менять данные которые использует другая транзакция. Результат выполнения параллельных транзакций должен быть таким, как если бы транзакции выполнялись последовательно.
- *Долговечность (Устойчивость).* После фиксации изменения не должны быть утеряны.

Транзакции классифицируются по признаку определения границ:

1) *Автоматические транзакции.* В этом режиме каждая инструкция T-SQL выполняется как отдельная транзакция. Если выполнение инструкции завершается успешно, происходит фиксация; в противном случае происходит откат.
2) *Неявные транзакции.* Если соединение работает в режиме неявных транзакции, то после фиксации или отката текущей транзакции автоматически начинает новую транзакцию. В этом режиме явно указывается только граница окончания транзакции с помощью инструкций COMMIT TRANSACTION и ROLLBACK TRANSACTION
3) *Явные транзакции.*

   Для определения явных транзакций используются следующие инструкции:

   - BEGIN TRANSACTION – задает начальную точку явной транзакции для соединения;
   - COMMIT TRANSACTION или COMMIT WORK – используется для успешного за-

вершения транзакции, если не возникла ошибка;

- ROLLBACK TRANSACTION или ROLLBACK WORK – используется для отмены транзакции, во время которой возникла ошибка.
- SAVE TRANSACTION – используется для установки точки сохранения или маркера внутри транзакции. Точка сохранения определяет место, к которому может возвратиться транзакция, если часть транзакции условно отменена. Если транзакция откатывается к точке сохранения, то ее выполнение должно быть продолжено до завершения с обработкой дополнительных инструкций языка T-SQL, если необходимо, и инструкции COMMIT TRANSACTION, либо транзакция должна быть полностью отменена откатом к началу. Для отмены всей транзакции следует использовать инструкцию ROLLBACK TRANSACTION; в этом случае отменяются все инструкции транзакции.

Управлением параллельным выполнением транзакции

Когда множество пользователей одновременно пытаются модифицировать данные в базе данных, необходимо создать систему управления, которая защитила бы модификации, выполняемым одним пользователем, от негативного воздействия модификаций, сделанных другими. Выделяют два типа управления параллельным выполнением:

- *Пессимистическое управление параллельным выполнением.* При пессимистическом подходе первый пользователь захвативший данные препятствует получению данных остальным. Если конфликты редки разумно выбрать оптимистическую стратегию, так как она обеспечивает более высокий уровень параллелизма.
- *Оптимистическое управление параллельным выполнением.* При оптимистическом подходе несколько пользователей получают в свое распоряжение копии данных. Первый завершивший редактирование сохраняет изменения, остальные же должны осуществить слияние изменений. Оптимистический алгоритм позволяет конфликту произойти, но система должна восстановиться после конфликта.

В случае пессимистичного управления в СУБД 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.**

Транзакция 1	Транзакция 2
UPDATE tbl1 SET f2=f2+20 WHERE f1=1;	UPDATE tbl1 SET f2=f2+25 WHERE f1=1;

В обеих транзакциях изменяется значение поля f2, по их завершении значение поля должно быть увеличено на 45. В действительности может возникнуть следующая последовательность действий: Обе транзакции одновременно читают текущее состояние поля. Точная физическая одновременность здесь не обязательна, достаточно, чтобы вторая по порядку операция чтения выполнилась до того, как другая транзакция запишет свой результат. Обе транзакции вычисляют новое значение поля, прибавляя, соответственно, 20 и 25 к ранее прочитанному значению. Транзакции пытаются записать результат вычислений обратно в поле f2. Поскольку физически одновременно две записи выполнить невозможно, в реальности одна из операций записи будет выполнена раньше, другая позже. При этом вторая операция записи перезапишет результат первой.

В результате значение поля f2 по завершении обеих транзакций может увеличиться не на 45, а на 20 или 25, то есть одна из изменяющих данные транзакций «пропадёт».

— **Проблема «грязного» чтения.** В транзакции 1 изменяется значение поля f2, а затем в

Транзакция 1	Транзакция 2
UPDATE tbl1 SET f2=f2+1 WHERE f1=1; ROLLBACK WORK;	SELECT f2 FROM tbl1 WHERE f1=1;

транзакции 2 выбирается значение этого поля. После этого происходит откат транзакции 1. В результате значение, полученное второй транзакцией, будет отличаться от значения, хранимого в базе данных.

— **Проблема неповторимого чтения.** Ситуация, когда при повторном чтении в рамках одной транзакции ранее прочитанные данные оказываются изменёнными.

Транзакция 1	Транзакция 2
UPDATE tbl1 SET f2=f2+3 WHERE f1=1; COMMIT;	SELECT f2 FROM tbl1 WHERE f1=1; SELECT f2 FROM tbl1 WHERE f1=1;

В транзакции 2 выбирается значение поля f2, затем в транзакции 1 изменяется значение поля f2. При повторной попытке выбора значения из поля f2 в транзакции 2 будет получен другой результат. Эта ситуация особенно неприемлема, когда данные считываются с целью их частичного изменения и обратной записи в базу данных.

— **Проблема чтения фантомов.** В транзакции 2 выполняется SQL-оператор, использу-

Транзакция 1	Транзакция 2
	SELECT f2 FROM tbl1 WHERE f1=1;
INSERT INTO tbl1 (f1,f2) VALUES (15,20); COMMIT;	
	SELECT f2 FROM tbl1 WHERE f1=1;

ющий все значения поля f2. Затем в транзакции 1 выполняется вставка новой строки, приводящая к тому, что повторное выполнение SQL-оператора в транзакции 2 выдаст другой результат. Такая ситуация называется чтением фантома (фантомным чтением). От не повторяющегося чтения оно отличается тем, что результат повторного обращения к данным изменился не из-за изменения/удаления самих этих данных, а из-за появления новых (фантомных) данных.

Для решение проблем используются следующие уровни изоляции:

- *Уровень 0* (read uncommitted) — запрещение «загрязнение» данных (повреждение данных). Этот уровень требует, чтобы изменять данные могла только одна транзакция, а другие транзакции ожидали завершение данной транзакции.
- *Уровень 1* (read committed) — запрещение «грязного» чтения. Если транзакция начала изменение данных, то никакая другая транзакция не сможет прочитать их до завершение первой.
- *Уровень 2* (repeatable read) — запрещение неповторяемого чтения. Если транзакция считывает данные, то никакая другая транзакция не сможет их изменить. Таким образом, при повторном чтении они будут находится в первоначальном состоянии.
- *Уровень 3* (snapshot, serializable)— 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 блокировки выполняется путем использования блокировок диапазона ключей. Подобная блокировка накладывается не на конкретные строки таблицы, а на строки удовлетворяющие определенному логическому условию.

При это одновременно может быть установлен только один параметр уровня изоляции, который продолжает действовать для текущего соединения до тех пор, пока не будет явно изменен. Уровни изоляции транзакции определяют тип блокировки, применяемый к операциям считывания. В любой момент транзакции можно переключиться с одного уровня изоляции на другой. Тогда

Уровень изоляции	Грязное чтение	Неповторяющееся чтение	Фантом
read uncommitted	Да	Да	Да
read committed	Нет	Да	Да
repeatable read	Нет	Нет	Да
snapshot	Нет	Нет	Нет
serializable	Нет	Нет	Нет

для транзакции изменяется уровень изоляции, ресурсы, которые считываются после изменения, защищаются в соответствии с правилами нового уровня. Ресурсы, которые считываются до изменения, остаются защищенными в соответствии с правилами предыдущего уровня.

Способы обработки ошибок. Если ошибка делает невозможным успешное выполнение транзакции, SQL Server автоматически выполняет ее откат и освобождает ресурсы, удерживаемые транзакцией. Если сетевое соединение клиента с SQL Server разорвано, то после того, как SQL Server получит уведомление от сети о разрыве соединения, выполняется откат всех необработанных транзакций для этого соединения. В случае сбоя клиентского приложения, выключения либо перезапуска клиентского компьютера соединение также будет разорвано, а SQL Server выполнит откат всех необработанных транзакций после получения уведомления о разрыве от сети. Если клиент выйдет из приложения, выполняется откат всех необработанных транзакций.

На случай возникновения ошибок код приложения должен содержать исправляющее действие: COMMIT или ROLLBACK. Эффективным средством для обработки ошибок, включая ошибки транзакций, является конструкция языка Transact-SQL TRY...CATCH

1.7 Блокировки. Определение, свойства, иерархии, гранулярность и взаимоблокировки, алгоритмы обнаружения взаимоблокировок

Определение

Блокировка — это механизм, с помощью которого компонент Database Engine синхронизирует одновременный доступ нескольких пользователей к одному фрагменту данных.

Свойства

1) гранулярностью (или размером блокировки)
2) режимом (или типом блокировки)
3) продолжительностью

1.7.1 Иерархии

Компонент Database Engine часто получает блокировки на нескольких уровнях гранулярности одновременно, чтобы полностью защитить ресурс. Такая группа блокировок на нескольких уровнях гранулярности называется иерархией блокировки. Например, чтобы полностью защитить операцию чтения индекса, экземпляру компоненту Database Engine может потребоваться получить разделяемые блокировки на строки и намеренные разделяемые блокировки на страницы и таблицу.

1.7.2 Гранулярность

Гранулярность блокировки определяет, какой ресурс блокируется в одной попытке блокировки.

Таблица 2

Ресурс	**Описание**
RID	Идентификатор строки, используемый для блокировки 1 строки в куче
KEY	Блокировка строки в индексе, используемая для защиты диапазонов значений ключа в сериализуемых транзакциях.
PAGE	8КБ страница в базе данных, например страница данных или индекса.
EXTENT	Упорядоченная группа из восьми страниц, например страниц данных или индекса.
HOBT	Куча или сбалансированное дерево. Блокировка, защищающая индекс или кучу стр. данных в таблице, не имеющей кластеризованного индекса.
TABLE	Таблица полностью, включая все данные и индексы.
FILE	Файл базы данных.
APPLICATION	Определяемый приложением ресурс.
METADATA	Блокировки метаданных.
ALLOCATION_UNIT	Единица размещения.
DATABASE	База данных, полностью.

1.7.3 Взаимоблокировки

Взаимоблокировки или тупиковые ситуации (deadlocks) возникают тогда, когда одна из транзакций не может завершить свои действия, поскольку вторая транзакция заблокировала нужные ей ресурсы, а вторая в то же время ожидает освобождения ресурсов первой транзакцией. Как SQL Server обнаруживает взаимоблокировки? Обнаружение взаимоблокировки вы-

полняется потоком диспетчера блокировок, который периодически производит поиск по всем задачам в экземпляре компонента Database Engine.

Следующие пункты описывают процесс поиска:

1) Значение интервала поиска по умолчанию составляет 5 секунд.
2) Если диспетчер блокировок находит взаимоблокировки, интервал обнаружения взаимоблокировок снижается с 5 секунд до 100 миллисекунд в зависимости от частоты взаимоблокировок.
3) Если поток диспетчера блокировки прекращает поиск взаимоблокировок, компонент Database Engine увеличивает интервал до 5 секунд.
4) Если взаимоблокировка была только что найдена, предполагается, что следующие потоки, которые должны ожидать блокировки, входят в цикл взаимоблокировки. Первая пара элементов, ожидающих блокировки, после того как взаимоблокировка была обнаружена, запускает поиск взаимоблокировок вместо того, чтобы ожидать следующий интервал обнаружения взаимоблокировки. Например, если текущее значение интервала равно 5 секунд и была обнаружена взаимоблокировка, следующий ожидающий блокировки элемент немедленно приводит в действие детектор взаимоблокировок. Если этот ожидающий блокировки элемент является частью взаимоблокировки, она будет обнаружена немедленно, а не во время следующего поиска взаимоблокировок.
5) Компонент Database Engine обычно выполняет только периодическое обнаружение взаимоблокировок. Так как число взаимоблокировок, произошедших в системе, обычно мало, периодическое обнаружение взаимоблокировок помогает сократить издержки от взаимоблокировок в системе.
6) Если монитор блокировок запускает поиск взаимоблокировок для определенного потока, он идентифицирует ресурс, ожидаемый потоком. После этого монитор блокировок находит владельцев определенного ресурса и рекурсивно продолжает поиск взаимоблокировок для этих потоков до тех пор, пока не найдет цикл. Цикл, определенный таким способом, формирует взаимоблокировку.
7) После обнаружения взаимоблокировки компонент Database Engine завершает взаимоблокировку, выбрав один из потоков в качестве жертвы взаимоблокировки. Компонент Database Engine прерывает выполняемый в данный момент пакет потока, производит откат транзакции жертвы взаимоблокировки и возвращает приложению ошибку 1205. Откат транзакции жертвы взаимоблокировки снимает все блокировки, удерживаемые транзакцией. Это позволяет транзакциям потоков разблокироваться, и продолжить выполнение. Ошибка 1205 жертвы взаимоблокировки записывает в журнал ошибок сведения обо всех

потоках и ресурсах, затронутых взаимоблокировкой.

1.8 Журнализация. Операции журнала транзакций и его логическая и физическая архитектуры. Модели восстановления. Метаданные

1.8.1 Типы используемых файлов

1) Первичные файлы данных.
2) Вторичные файлы данных.
3) Файлы журналов.

1.8.2 Операции

1) Восстановление отдельных транзакций.
2) Восстановление всех незавершенных транзакци при запуске SQL Server.
3) Накат восстановленно базы данных, файла, файлово группы или страницы до момента сбоев.
4) Поддержка репликации транзакций.
5) Поддержка решений с резервными серверами.

1.8.3 Логическая архитектура

На логическом уровне журнал транзакций состоит из последовательности записей. Двумя основными типами записи Log-файла являются:

1) код выполненной логической операции.
   - Для наката логической операции выполняется эта операция.
   - Для отката логической операции выполняется логическая операция, обратная зарегистрированной.
2) исходны и результирующий образ измененных данных.
   - Для наката операции применяется результирующий образ.
   - Для отката операции применяется исходный образ.

1.8.4 Физическая архитектура

Рисунок 2 – B-Tree

1.8.5 Модель восстановления

Модель восстановления — это свойство базы данных, которое управляет процессом регистрации транзакций, определяет, требуется ли для журнала транзакций резервное копирование, а также определяет, какие типы операций восстановления доступны.

1) Модель полного восстановления (FULL).
2) Модель восстановления с неполным протоколированием (BULK_LOGGED).
3) Простая модель восстановления (SIMPLE).

Восстановление данных

1) База данных (полное восстановление базы данных).
2) Файл данных (восстановление файла).
3) Страница данных (восстановление страницы).

Метаданные?

Метаданные, в общем случае, это данные о данных, информация об информации, описание контента.

???

1.9 Безопасность и Аудит. Ключевые понятия и участники системы безопасности. Модели управления доступом

Ждет своего часа

1.10 MPP системы. Распределенное и колоночное хранение. Распределенные вычисления, модель MapReduce. Обеспечение отказоустойчивости.

Ждет своего часа

1.11 In-Memory базы данных. Преимущества и недостатки. Примеры использования

Ждет своего часа

1.12 Инструкции языка описания данных, инструкции языка обработки данных, инструкции безопасности, инструкции управления транзакциями

Инструкции языка описания данных

Описание данных: **Data Definition Language (DDL)** – это группа операторов описания данных. Другими словами, с помощью операторов, входящих в эту группы, мы определяем структуру базы данных и работаем с объектами этой базы, т. е. создаем, изменяем и удаляем их.

- CREATE — создает объект базы данных.
- ALTER — изменяет объект базы данных.
- DROP — удаляет объект базы данных.
- ENABLE TRIGGER — включает триггер DML, DDL или logon.
- DISABLE TRIGGER — отключает триггер.
- TRUNCATE TABLE — удаляет все строки в таблице, не записывая в журнал удаление отдельных строк.
- UPDATE STATISTICS — обновляет статистику оптимизации запросов для таблицы или индексированного представления.

Инструкции языка обработки данных

Обработка данных: **Data Manipulation Language (DML)** – это группа операторов для манипуляции данными. С помощью этих операторов мы можем добавлять, изменять, удалять и выгружать данные из базы, т. е. манипулировать ими.

1) SELECT — читывает данные, удовлетворяющие заданным условиям.
2) INSERT — обавляет новые данные.
3) UPDATE — изменяет существующие данные.
4) DELETE — удаляет данные.
5) MERGE — выполняет операции вставки, обновления или удаления для целевой таблицы на основе результатов соединения с исходной таблицей. Если условие, по которому происходит объединение — истина, то можно выполнить обновление или удаление, иначе — вставку.

```
1          MERGE Основная< таблиц>
2          USING Таблица< или запрос источника>
3          ON Условия< объединения>
```

```
4          [ WHEN MATCHED [ AND Доп<. условие> ]]
5          THEN < UPDATE и л и DELETE >
6          [ WHEN NOT MATCHED [ AND Доп. условие> ]
7          THEN < INSERT > ]
8          [ WHEN NOT MATCHED BY SOURCE [ AND Доп<. условие> ]
9          THEN < UPDATE или DELETE > ] [ ... n ]
10         [ OUTPUT ]
11         ;
12
```

6) BULK INSERT — полняет импорт файла данных в таблицу или представление базы данных в формате, указанном пользователем.
7) READTEXT — итывает значения text, ntext или image из столбцов типа text, ntext или image начиная с указанной позиции
8) WRITETEXT — обновляет и заменяет все поле text, ntext или image Базы данных
9) UPDATETEXT — обновляет часть поля text, ntext или imag

```
1    READTEXT { table . column text_ptr offset size } [ HOLDLOCK ]
2
3    UPDATETEXT [ BULK ] { table_name . dest_column_name dest_text_ptr }
4    { NULL | insert_offset }
5    { NULL | delete_length }
6    [ WITH LOG ]
7    [ inserted_data
8    | { table_name . src_column_name src_text_ptr } ]
9
10   WRITETEXT [ BULK ]
11   { table . column text_ptr }
12   [ WITH LOG ] { data }
```

Инструкции безопасности

Инструкции безопасности (ранее: доступа к данным): **Data Control Language (DCL)**.

SQL Server обеспечивает защиту данных от неавторизированного доступа и от фальсификации. Основными функциями безопасности SQL Server являются:

- проверка подлинности (аутентификация) – процедура проверки соответствия некоего лица и его учетной записи в компьютерной системе. Один из способов аутентификации состоит в задании пользовательского идентификатора, в просторечии называемого «логином» (login – регистрационное имя пользователя) и пароля – некой конфиденциальной информации, знание которой обеспечивает владение определенным ресурсом.
- авторизация – это предоставление лицу прав на какие-то действия в системе. В основе системы безопасности SQL Server лежат три понятия:

В основе системы безопасности SQL Server лежат три понятия:

- участники системы безопасности (Principals);
- защищаемые объекты (Securables);
- система разрешений (Permissions).

Участники системы безопасности или принципалы – это сущности, которые могут запрашивать ресурсы SQL Server.

Защищаемые объекты – это ресурсы, доступ к которым регулируется системой авторизации.

Инструкции безопасности:

- GRANT — предоставляет пользователю разрешения на определенные операции с объектом.
- REVOKE — отзывает ранее выданные разрешения.
- DENY — задает запрет, имеющий приоритет над разрешением.
- ADD SIGNATURE — добавляет цифровую подпись для хранимой процедуры, функции, сборки или триггера.
- OPEN MASTER KEY — открывает главный ключ в текущей базе данных.
- CLOSE MASTER KEY — закрывает главный ключ в текущей базе данных.
- OPEN SYMMETRIC KEY — расшифровывает симметричный ключ и делает его доступным для использования.
- CLOSE SYMMETRIC KEY — закрывает симметричный ключ или все симметричные ключи, открытые в текущем сеансе.
- EXECUTE AS — контекст выполнения сеанса переключается на заданное имя входа и имя пользователя.

- REVERT — переключает контекст выполнения в контекст участника, вызывавшего последнюю инструкцию EXECUTE AS.
- SETUSER — позволяет члену предопределенной роли сервера sysadmin или члену предопределенной роли базы db_owner олицетворять другого пользователя.

Инструкции управления транзакциями

Transaction Control Language (TCL) – группа операторов для управления транзакциями.

Транзакция – это команда или блок команд (инструкций), которые успешно завершаются как единое целое, при этом в базе данных все внесенные изменения фиксируются на постоянной основе или отменяются, т. е. все изменения, внесенные любой командой, входящей в транзакцию, будут отменены.

- BEGIN DISTRIBUTED TRANSACTION — запускает распределенную транзакцию, управляемую координатором распределенных транзакций.
- BEGIN TRANSACTION — отмечает начальную точку явной локальной транзакции.
- COMMIT TRANSACTION — отмечает успешное завершение явной или неявной транзакции.
- COMMIT WORK — действует так же, как и инструкция COMMIT TRANSACTION.
- ROLLBACK TRANSACTION — откатывает явные или неявные транзакции до начала или до точки сохранения транзакции.
- ROLLBACK WORK — действует так же, как и инструкция ROLLBACK TRANSACTION.
- SAVE TRANSACTION — устанавливает точку сохранения внутри транзакции.

1.13 Объекты базы данных: функции, процедуры, триггеры и курсоры

1.14 Оптимизация запроса: индексы, партиционирование, сегментирование

1.14.1 Индексы

(материал на основе SQL Server)

Индекс – это объект базы данных, обеспечивающий дополнительные способы быстрого поиска и извлечения данных. Индекс может создаваться на одном или нескольких столбцах. Это означает, что индексы бывают простыми и составными. Если в таблице нет индекса, то поиск нужных строк выполняется простым сканированием по всей таблице. При наличии индекса время поиска нужных строк можно существенно уменьшить.

К недостаткам индексов следует отнести:

- дополнительное место на диске и в оперативной памяти,
- замедляются операции вставки, обновления и удаления записей.

В SQL Server (и, наверное, многих других СУБД) индексы хранятся в виде сбалансирован-

ных деревьев. Представление индекса в виде сбалансированного дерева означает, что стоимость поиска любой строки остается относительно постоянной, независимо от того, где находится эта строка.

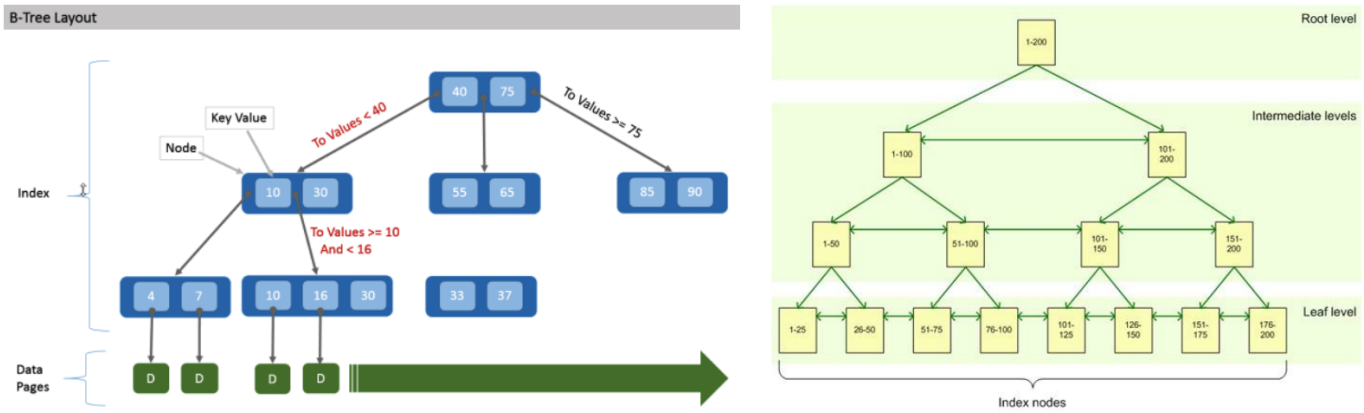

Рисунок 3 – B-Tree

Существует два типа индексов:

- Кластерные индексы
- Некластерные индексы, которые включают:
  - индексы на основе кучи;
  - индексы на основе кластерных таблиц.

Кластерные индексы

В кластерном индексе таблица представляет собой часть индекса, или индекс представляет собой часть таблицы в зависимости от вашей точки зрения. Листовой узел кластерного индекса – это страница таблицы с данными. Поскольку сами данные таблицы являются частью индекса, для таблицы может быть создан только один кластерный индекс. Кластерный индекс является уникальным индексом по определению.

Некластерные индексы на основе кучи

В листьях некластерного индекса на основе кучи хранятся указатели на строки данных. Указатель строится на основе идентификатора файла (ID), номера страницы и номера строки на странице. Весь указатель целиком называется идентификатором строки (RID).

Некластерные индексы, основанные на кластерных таблицах

В листьях некластерного индекса, основанного на кластерных таблицах, хранятся указатели на корневые узлы кластерных индексов. Поиск в таком индексе состоит из двух этапов:

- поиск в некластерном индексе;
- поиск в кластерном индексе.

Создание индекса

В SQL Server индексы создаются при помощи команды **CREATE INDEX**.

Помимо этого, индексы создаются при добавлении некоторых ограничений. Такой тип индексов часто называют «связанными индексами». Связанные индексы создаются при добавлении одного из следующих двух типов ограничений:

- ограничения первичного ключа (PRIMARY KEY);
- ограничения уникальности (UNIQUE).

1.14.2 Партиционирование

(материал на основе PostgreSQL)

Партиционирование – это метод разделения больших (исходя из количества записей, а не столбцов) таблиц на много маленьких.

Для произведения партиционирования нужно решить, каким будет ключ партиционирования – другими словами, по какому алгоритму будут выбираться партиции. Есть пара наиболее очевидных:

1) партиционирование по дате – например, выбирать партиции, основываясь на годе, в котором пользователь был создан.

   Достоинства:

   – легко понять;

   – количество строк в данной таблице будет достаточно стабильным.

   Недостатки:

   – требует поддержки – время от времени нам придётся добавлять новые партиции;

   – поиск по имени пользователя или id потребует сканирования всех партиций;

2) партиционирование по диапазону идентификаторов – например, первый миллион пользователей, второй миллион пользователей, и так далее

   Достоинства:

   – легко понять;

   – количество строк в данной таблице будет абсолютно стабильным.

   Недостатки:

   – требует поддержки – время от времени нам придётся добавлять новые партиции;

   – поиск по имени пользователя потребует сканирования всех партиций;

3) партиционирование по чему-нибудь другому – например, по первой букве имени пользователя.

   Достоинства:

   – легко понять;

   – никакой поддержки — есть строго определенный набор партиций и нам никогда не

придется добавлять новые;

Недостатки:

- количество строк в партициях будет стабильно расти;
- в некоторых партициях будет существенно больше строк, чем в других (больше людей с никами, начинающимися на «t*», чем на «y*»);
- поиск по id потребует сканирования всех партиций;

4) есть еще пара других, не так часто используемых вариантов, вроде «партиционирования по хэшу от имени пользователя».

Пример

```
1   CREATE TABLE measurement (
2   city_id int not null,
3   logdate date not null,
4   peaktemp int,
5   unitsales int
6   ) PARTITION BY RANGE (logdate);
```

1.14.3 Сегментирование

(материал на основе Vertica)

Для распределения данных по кластерам используется их сегментация(Segmentation), а точнее сегментация проекций(Projection) в которых они находятся. Задача разработчика — подобрать такой список полей и/или такую функцию(например, хэш-функцию), благодаря которым данные равномерно распределятся по нодам кластера. HP Vertica рекомендует использовать встроенные функции HASH и MODULARHASH для этих целей.

Прим. Для справки: *K-Safety* — критерий отказоустойчивости БД, определяющий без какого кол-ва узлов БД сможет функционировать корректно. Он может быть равен 0, 1 или 2.

Для обеспечения K-Safety > 0 создаются сообщные проекции(Buddy Projection). Проекции называются сообщными, если они имеют в себе одинаковые наборы полей и одинаковое выражение сегментации, но хранятся на разных нодах. Сообщные проекции позволяют создать те самые реплики, которые позволяют работать БД в выбранном режиме K-Safety.

Пример

1.15 План запроса. Этапы выполнения запроса

Ждет своего часа.

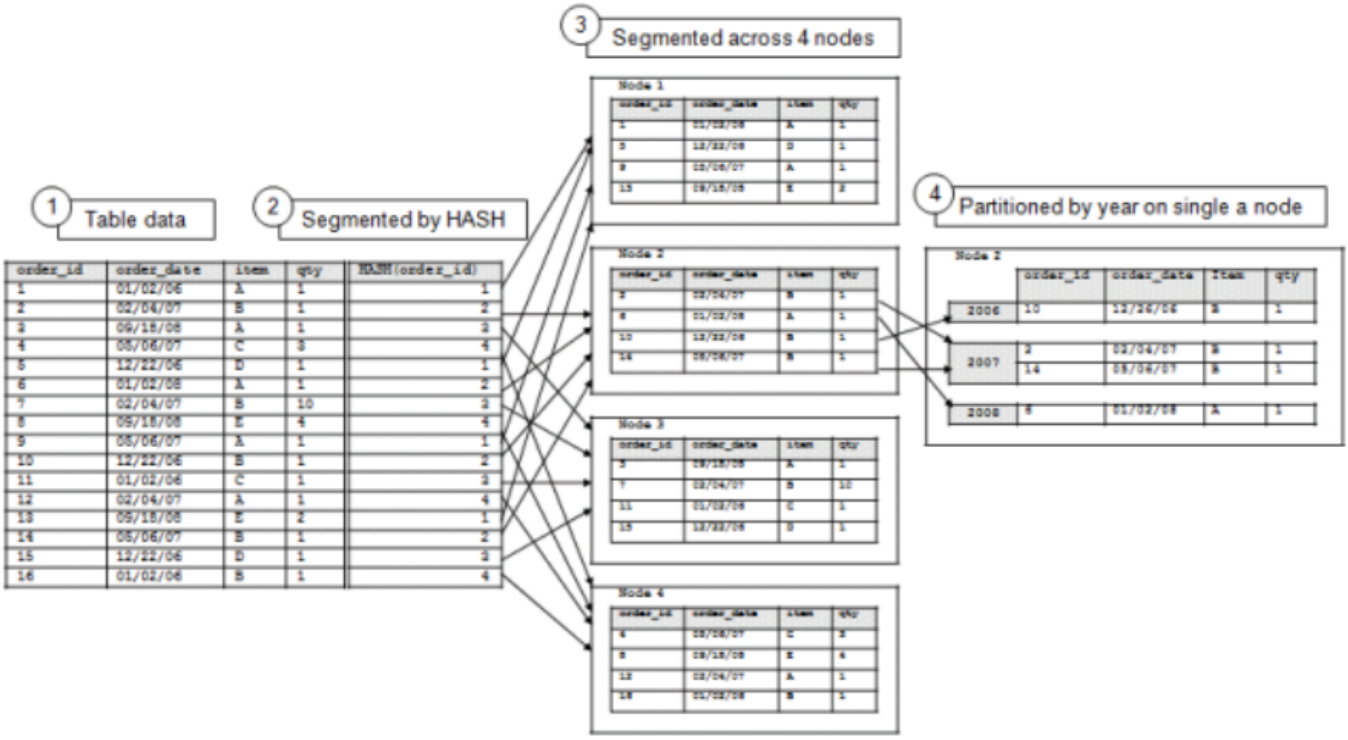

Рисунок 4 – Сегментирование и партиционирование вместе