1. EER-моделирование баз данных. Суперклассы и подклассы. Обоснование целесообразности определения подклассов и суперклассов. Специализация и генерализация. Ограничения полноты и непересечения. Примеры.

Понятий, рассмотренных ранее, достаточно для представления многих прикладных баз данных, которые в основном встречаются в экономике и производстве. Однако начиная с некоторого момента, стали появляться новые приложения технологии баз данных: инженерное проектирование, образные и графические БД, картографические и геологические, многоаспектные БД (представляющие как традиционные, так и неструктурированные данные: текст, рисунки, аудио- и видеозаписи и пр.) и основы знаний для приложений искусственного интеллекта. Эти типы баз данных имеют более сложные требования, чем традиционные. Для того чтобы представить эти требования наиболее точно, требуется, как минимум, развить «семантику» понятий моделирования. Первым этапом этого развития может служить расширение ранее рассмотренной ER-модели путём включения в неё новых понятий, позволяющих более полно и детально отразить специфику предметной области. Этот расширенный вариант ER-модели назовём EER- (продвинутой ER-) моделью.

Например: члены типа объекта **СЛУЖАЩИЙ** могут быть сгруппированы на **СЕКРЕТАРЕЙ, ИНЖЕНЕРОВ, УПРАВЛЯЮЩИХ, ТЕХНИКОВ** и т. п. Множество объектов в каждой из групп является подмножеством объектов, принадлежащих типу объекта **СЛУЖАЩИЙ,** что означает, что каждый объект, являющийся членом одной из этих подгрупп, также является **СЛУЖАЩИМ**. Каждую из подгрупп назовем ***подклассом*** типа объекта **СЛУЖАЩИЙ,** а сам **СЛУЖАЩИЙ** называется ***суперклассом*** каждого из этих подклассов.

Суперкласс – тип сущности (объекта), включающий в себя различные вспомогательные группы экземпляров, которые обладают собственной спецификой, и эти группы необходимо представить в модели данных.

Подкласс – различимая вспомогательная группировка экземпляров типа сущности, исполняющая в нём отдельную роль, и которая должна быть представлена в модели данных.

Подкласс **наследует** все атрибуты своего суперкласса. Объект подкласса наследует также все экземпляры связей для типов связей, в которых участвует суперкласс. Кроме того, подкласс может иметь свои специфические атрибуты и самостоятельно участвовать в отдельных связях.

Специализация (нисходящий подход).

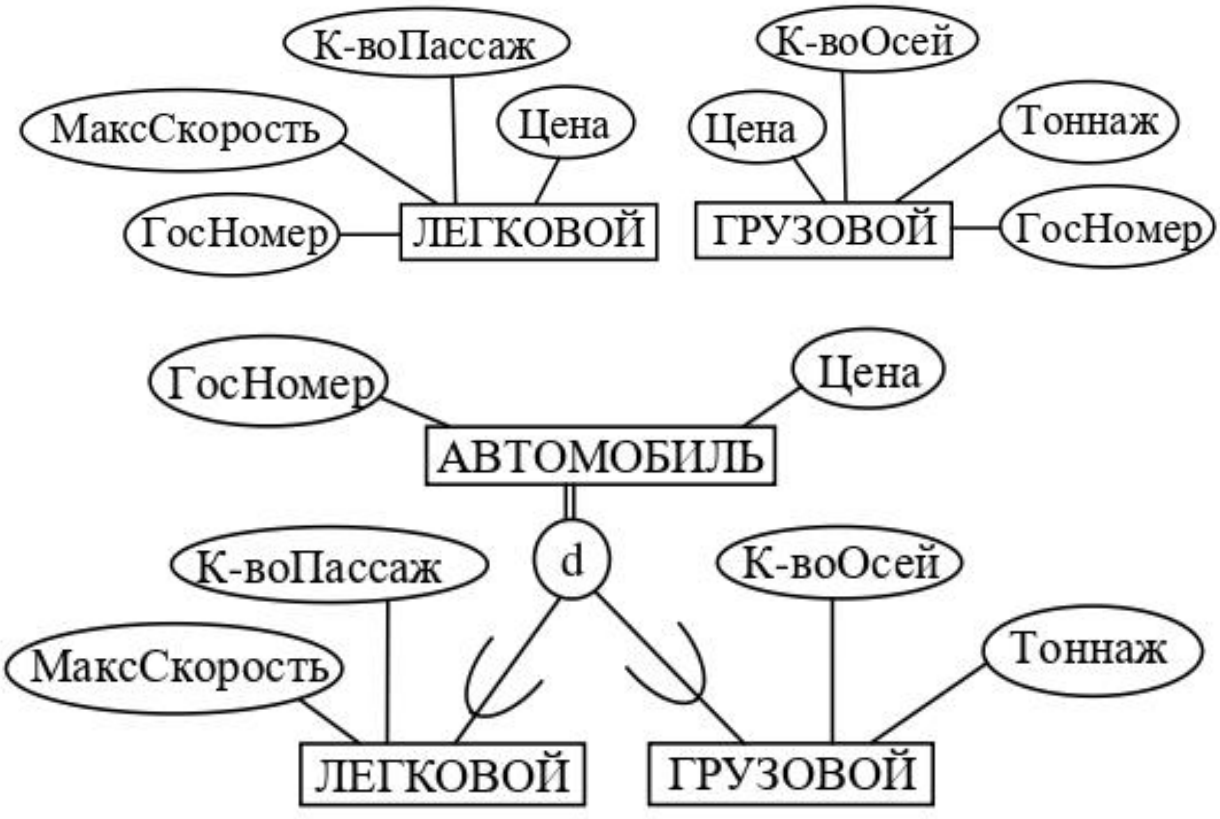
Процесс выявления различий между отдельными экземплярами типа сущности за счёт выделения их дополнительных характеристик (процесс определения множества подклассов типа объекта-суперкласса).

Множество подклассов, формируемых специализацией, определяется на основе некоторых характеристик, выделяющих объект в суперклассе, и выражающих определенные различия среди объектов. Например, множество подклассов **{СЕКРЕТАРЬ, ИНЖЕНЕР, ТЕХНИК} –** специализация суперкласса **СЛУЖАЩИЙ**, основанная на *форме работы* каждого служащего.

Может существовать несколько специализаций одного и того же типа объекта, основанных на различных характеристиках. Например: другой специализацией **СЛУЖАЩЕГО** являются подклассы **{СЛУЖАЩИЙ-НА-ОКЛАДЕ, СЛУЖАЩИЙ-ПОЧАСОВИК};** эта специализация выделяется *формой оплаты*.

Генерализация (обобщение) – восходящий подход.

Процесс сведения к минимуму различия между типами сущностей, путём выделения их общих характеристик. Этот процесс приводит к созданию обобщённого суперкласса на основе первоначальных типов сущностей (которые становятся подклассами).

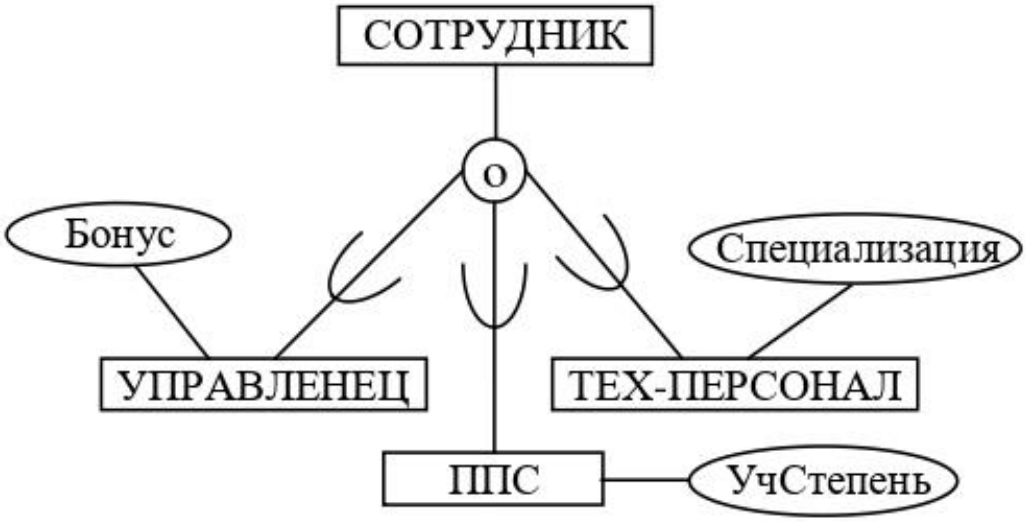


Процесс генерализации можно представлять как обратный к процессу специализации. Можно рассматривать **{ЛЕГКОВОЙ, ГРУЗОВОЙ}** как специализацию типа объекта **АВТОМОБИЛЬ,** а **АВТОМОБИЛЬ** как генерализацию типов объектов **ЛЕГКОВОЙ** и **ГРУЗОВОЙ**.

В процессе специализации/генерализации могут применяться ограничения двух типов: **ограничения непересечения** и **ограничения полноты**.

Ограничение непересечения означает, что подклассы не имеют общих членов, т. е. объект может быть членом *не более чем одного* подкласса.

Пример: подклассы **{НА_ОКЛАДЕ, ПОЧАСОВИК}**.



Если подклассы *НЕ* непересекающиеся, то их множества объектов *могут пересекаться*, т. е. один и тот же объект может быть членом более чем одного подкласса.

Ограничение полноты (степени участия) определяет, должен ли быть отнесён к какому-либо подклассу каждый элемент суперкласса.

Полное участие означает, что каждый экземпляр суперкласса должен быть членом хотя бы одного подкласса *(двойная линия при суперклассе)*.

Частичное участие допускает ситуацию, когда объект суперкласса не принадлежит ни одному из подклассов *(одинарная линия при суперклассе)*.

2. Категоризация, категории. Примеры. Возможность представления категоризации как специализации/генерализации. Иерархии и решётки.

Категоризация.

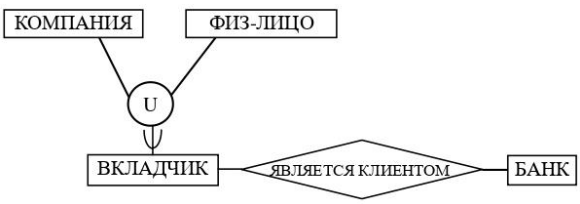
В некоторых случаях есть необходимость смоделировать единственную суперкласс/подкласс связь с *более чем одним* суперклассом, где суперклассы представляют различные типы объектов. В этом случае подкласс называется категорией.

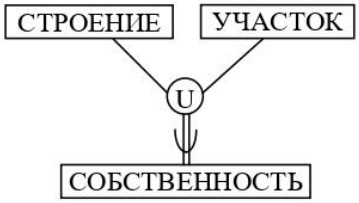
Пример. Имеется три типа объекта: ЛИЦО, БАНК, КОМПАНИЯ. Каждый из этих объектов может быть собственником СРЕДСТВА ПЕРЕДВИЖЕНИЯ. Необходимо образовать класс, включающий

экземпляры всех трёх типов объектов, играющих роль собственника средства передвижения. С этой целью создадим категорию СОБСТВЕННИК как подкласс, *объединяющий* три класса.

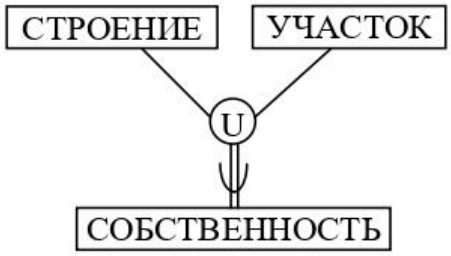
Категория - подмножество *объединения* суперклассов. Следовательно, объект, являющийся членом категории **СОБСТВЕННИК**, должен существовать, *по крайней мере, в одном* из суперклассов, но не обязан быть членом *всех* из них. Это означает то ограничение, что **СОБСТВЕННИК** может быть либо **КОМПАНИЕЙ,** либо **БАНКОМ** либо **ЛИЦОМ**.

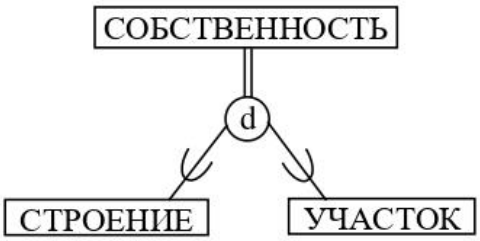
Каждый экземпляр объекта **СОБСТВЕННИК** наследует атрибуты либо **КОМПАНИИ,** либо **ЛИЦА,** либо **БАНКА** в зависимости от суперкласса, к которому данный объект принадлежит. Это называется *выборочной наследственностью*. Однако бывают ситуации, когда подклассы-категории наследуют атрибуты *всех* своих суперклассов.





Категория может быть *полной* или *частичной***.** Например: **ВКЛАДЧИК** – частичная категория (одинарная линия), поскольку не каждая **КОМПАНИЯ** и не каждое **ФИЗ-ЛИЦО** обязано являться вкладчиком банка. Однако, категория **СОБСТВЕННОСТЬ** является полной, так как каждое **СТРОЕНИЕ** и каждый **УЧАСТОК** должны быть её членами (двойная линия). Суперклассы подкласса объединения могут иметь различные ключи (домены), как **СОБСТВЕННИК** или одинаковые, как **СРЕДСТВО ПЕРЕДВИЖЕНИЯ**.





В случае полной (не частичной) категории она может быть представлена как специализация или генерализация, а выбор формы представления субъективен. Если два класса представляют один и тот же тип объекта с общими многочисленными атрибутами, включая общие ключи, предпочтительнее генерализация; в противном случае – категоризация.

3. Варианты отображения EER-модели в реляционную модель. Примеры.

Пусть имеется суперкласс C с атрибутами K, A_1, A_2, …, A_n (K – ключ) и m подклассов S_1, S_2, …, S_m с множествами своих специфических атрибутов

$A(S_1)$, $A(S_2)$, …, $A(S_m)$. Преобразование в реляционную схему допускает
4 варианта:

Вариант 1.

Для C создаётся таблица $R(K, A_1, A_2, …, A_n)$. Для каждого S_i создаётся таблица R_i, включающая специфические атрибуты R_i и первичный ключ суперкласса C: $R_i(K, A(S_i))$. При этом K – ключ как в R, так и в каждом R_i. Операции соединения R_i и R по ключу K обеспечивают полный комплект атрибутов для каждого S_i (и наследованных, и специфических).
Вариант работает для любых ограничений на подклассы: пересекающихся и непересекающихся, с полным участием или с частичным.

Вариант 2.

Для суперкласса C таблица не создаётся. Для каждого S_i создаётся таблица R_i, включающая как наследованные, и специфические атрибуты подкласса S_i, и ключ K: $R_i(K, A_1, A_2, …, A_n, A(S_i))$. При этом K является ключом в каждом R_i. Полный комплект атрибутов каждого подкласса S_i обеспечивается изначально, без необходимости соединения. Однако для получения всех объектов суперкласса C придётся воспользоваться внешним соединением.
Вариант хорошо работает только при ограничениях полноты и непересечения. Если нет полноты, то теряются объекты, не принадлежащие ни к одному подклассу, а в случае перекрытия – дублируются объекты, принадлежащие нескольким подклассам с одновременным дублированием всех унаследованных атрибутов и со всеми вытекающими последствиями.
Так как приходится часто просматривать все таблицы R_i, этот вариант обычно не рекомендуется.

Вариант 3.

Создаётся только одна таблица R, включающая как атрибуты суперкласса, так и всех подклассов: $R(K, A_1, A_2, …, A_n, A(S_1), A(S_2), …, A(S_m), t)$. Кроме того, в схеме R присутствует атрибут t, указывающий на подкласс, к которому относится объект. Областью определения t может быть {1, 2, …, m}. В случае частичного участия добавляется значение NULL.
Этот вариант только для непересекающихся подклассов.

Вариант 4.

Создаётся также только одна таблица R, которая включает атрибуты суперкласса, атрибуты подклассов и m атрибутов-индикаторов t_1, t_2, …, t_m: $R(K, A_1, A_2, …, A_n, A(S_1), A(S_2), …, A(S_m), t_1, t_2, …, t_m)$. Каждое t_i – это есть булевская переменная, указывающая на принадлежность к подклассу S_i.
Это вариант для перекрывающихся подклассов.

Варианты 3 и 4 рекомендуются, если специфических атрибутов немного, а операции соединения проводить не надо, что приводит к повышению эффективности.

4. Аномалии обновления в реляционных таблицах. Примеры. Функциональные зависимости и их детерминанты. Понятие о

нормализации реляционных таблиц. Полные и частичные зависимости. 1-я и 2-я нормальные формы и порядок к ним приведения.

На этапе реляционного проектирования базы данных возникает задача о составе реляционных таблиц.

Следует сгруппировать атрибуты объектов (и связей) в таблицы так, чтобы минимизировать избыточность данных и тем самым сократить объём памяти, необходимый для физического хранения данных в таблицах.

В случае наличия избыточности работа с базой данных может быть сопряжена с рядом аномалий обновления.

Аномалии вставки:

1. При вставке сведений о новом сотруднике необходимо указывать и сведения о его отделе, которые должны (побуквенно) соответствовать сведениям об этом же отделе в других строках таблицы Сотрудники. И это соответствие необходимо всякий раз отслеживать, что достаточно неудобно.
2. При вставке сведений о новом отделе, который ещё не имеет собственных сотрудников, потребуется присвоить значение Null всем атрибутам описания сотрудников, включая ключевой таб. номер. Что вызовет нарушение целостности сущностей, и будет системой отвергнуто.
3. **Аномалии модификации:**
4. При попытке изменения значения одного из атрибутов отделов (например, номера телефона отдела «физмат») необходимо обновить соответствующие значения для всех сотрудников этого отдела. Если при этом какой-либо сотрудник будет пропущен, то база данных будет в итоге содержать противоречивые сведения.
5. **Аномалии удаления:**
6. При удалении из таблицы Сотрудники строки с информацией о последнем сотруднике некоторого отдела (например, об Алексееве), сведения об этом отделе будут полностью удалены из базы данных.

Функциональные зависимости

Пусть:

R – реляционная таблица,
A(R) – её схема (набор атрибутов),
A, B – группы атрибутов R: $A \subset A(R)$, $B \subset A(R)$.

Если в таблице R совокупное значение группы атрибутов A может появляться в связке только с одним совокупным значением группы атрибутов B, то говорят, что в таблице R имеется **функциональная зависимость** B от A:

$$f: A \rightarrow B$$

A называется **детерминантом** функциональной зависимости f.
Функциональная зависимость – понятие смысловое (семантическое). Определяется спецификой предметной области и ограничениями, наложенными на некоторые атрибуты.

Например:

1. От ключевых атрибутов в таблице функционально зависят все остальные.
2. Если $B \subset A$, то, очевидно, $A \rightarrow B$.

Зависимости типа 2 называются **тривиальными**. Они не несут никакой полезной информации, и рассматриваться не будут.

По этой же причине если $A \to B$ и $A \cap B \neq \varnothing$, то зависимость $A \to B$ может быть сведена к зависимости $A \to B - A$.

В связи с этими соображениями будут рассматриваться только такие зависимости $A \to B$, где A и B не пересекаются.

Любые аномалии обновления связаны с существованием «лишних» функциональных зависимостей в реляционных таблицах. В идеале следует стремиться к тому, чтобы зависимости были только от ключей. **Нормализация** – это и есть способ избавления от «лишних» зависимостей. Она заключается в приведении таблиц к той или иной **нормальной форме (НФ)**: от низших к более высшим.

Каждая следующая нормальная форма ограничивает определённый вид функциональных зависимостей, снижает избыточность данных, устраняет соответствующие аномалии обновления и сохраняет свойства предыдущих нормальных форм.

Первая нормальная форма (1НФ)

Основное требование: таблицы должны соответствовать реляционной модели данных, с соблюдением определённых реляционных принципов.

А именно:

- В таблице не должно быть дублирующих строк;
- В столбце должны храниться данные одного типа (или более усиленно: одного домена);
- В каждой ячейке таблицы должно храниться атомарное значение.

Правило приведения таблицы к 1НФ: Если у объекта имеется многозначный (по смыслу) атрибут, то он перемещается в отдельную таблицу, вместе с копией первичного ключа. Тогда избыточность, связанная с этим атрибутом, будет минимально возможной.

Пример.

ФИО	Телефон	Тип телефона
Иванов С. Б.	(495)123-45-67	
Иванов С. Б.	(499)321-54-76	
Петров Д. Д.	(499)223-45-54	Домашний
Петров Д. Д.	(916)112-33-55	Мобильный
Сидоров В. Н.	(905)333-55-77	

Вторая нормальная форма (2НФ)

Пусть в таблице R имеется зависимость f: $A \to B$.

Если существует такое $A_1 \subset A$ и $A_1 \neq A$, что при переходе от A к A_1 зависимость f сохраняется, (т. е. имеет место f: $A_1 \to B$), то говорят, что f – **частичная зависимость**.

В противном случае, т. е. если удаление из A хотя бы одного атрибута ведёт к утрате функциональной зависимости, говорят, что f – **полная зависимость**.

Требование 2НФ: реляционная таблица находится в 2НФ, если она находится в 1НФ и, кроме того, каждый неключевой атрибут полно зависит от первичного ключа.

Порядок приведения таблицы к 2НФ:

Если имеется частичная зависимость: $A \rightarrow B$ и $A_1 \rightarrow B$, где $A_1 \subset A$ и $A_1 \neq A$, то атрибуты группы B перемещаются в новую отдельную таблицу, вместе с копией атрибутов группы A_1. При этом в этой новой таблице A_1 становится ключом.

5. Транзитивные зависимости в реляционных таблицах. 3-я нормальная форма, нормальная форма Бойса-Кодда и порядок к ним приведения. Примеры.

Третья нормальная форма (3НФ)

Если в таблице R имеются зависимости $A \rightarrow B$ и $B \rightarrow C$, то говорят, что C **транзитивно зависит** от A.

Требование 3НФ: реляционная таблица находится в 3НФ, если она находится во 2НФ и не имеет неключевых атрибутов, транзитивно зависящих от первичного ключа.

Порядок приведения таблицы к 3НФ.

Пусть K – первичный ключ таблицы R. Если в R имеется транзитивная зависимость $K \rightarrow B \rightarrow C$, то атрибуты группы C перемещаются в новую отдельную таблицу вместе с копией атрибутов группы B. При этом B становится ключом в новой таблице.

Пример. Таблица СЛ2(Код, Фамилия, Имя, …, *No*, НазвОтд, …)

Функциональные зависимости:

f_1: Код $\rightarrow$ Фамилия, Имя, …, *No*, НазвОтд, …

Зависимость от первичного ключа.

f_2: *No* $\rightarrow$ НазвОтд

Зависимость от неключевого атрибута.

Нормализованный вариант:

Таблицы:

СЛ(*Код,* Фамилия, Имя, …, *No*, …), ОТ(No, НазвОтд, …)

Нормальная форма Бойса-Кодда (НФБК, усиленная 3-я)

Требование НФБК: реляционная таблица находится в НФБК тогда и только тогда, когда детерминант любой её функциональной зависимости является потенциальным ключом (суперключом).

При условии соблюдения в таблице R требований 3НФ нарушение требований НФБК может быть только в следующем случае:

- имеются два или более составных потенциальных ключа таблицы R;
- эти составные ключи перекрываются (т. е. совместно используют хотя бы один атрибут).

Порядок приведения таблицы к НФБК.

- Если в таблице R имеется зависимость $A \rightarrow B$, где A не является потенциальным ключом, то атрибуты группы B перемещаются в отдельную таблицу вместе с копией атрибутов группы A.

Пример.

Наша проектная организация проводит собеседования с клиентами (заказчиками).

Условия:

1. Сотруднику организации для беседы с клиентами предоставляется комната – **только одна в день**. Но в течение дня эта же комната может использоваться и другими сотрудниками.
2. С каждым конкретным клиентом может проводиться **только одно собеседование в день**. Но в другой день собеседование с ним также может проводиться.

Для отображения фактов собеседования имеется таблица:

Собеседование(Код-сотр, Код-кл, Дата, Время, Комната)

6. Многозначные зависимости в реляционных таблицах. 4-я нормальная форма и порядок к ней приведения. Примеры.

Четвёртая нормальная форма (4НФ)

Многозначная зависимость.

Пусть: R – реляционная таблица,

A(R) – её схема (набор атрибутов),

A, B, C – группы атрибутов R: $A \subset A(R)$, $B \subset A(R)$, $C \subset A(R)$

Если в таблице R совокупное значение группы атрибутов A может появляться в связке только с определённым множеством совокупных значений группы атрибутов B, и это множество совершенно не зависит от группы C, то говорят, что в таблице R имеется **многозначная зависимость** B от A (из A в B):

f: $A \twoheadrightarrow B$

Можно показать, что если имеется многозначная зависимость $A \twoheadrightarrow B$, то также имеется многозначная зависимость $A \twoheadrightarrow C$.

Поэтому иногда пишут: $A \twoheadrightarrow B \mid C$

Многозначная зависимость $A \twoheadrightarrow B$ называется **тривиальной**, если $B \subset A$ или $A \cup B = A(R)$. В противном случае – **нетривиальной**.

Требование 4НФ: реляционная таблица находится в 4НФ тогда и только тогда, когда она находится в НФБК и не содержит нетривиальных многозначных зависимостей.

Порядок приведения таблицы к 4НФ.

Если в таблице имеется нетривиальная многозначная зависимость $A \twoheadrightarrow B$, то атрибуты группы B перемещаются в новую отдельную таблицу, вместе с копией атрибутов группы A.

Пример.

Курсы (уч. дисциплины); ***Преподаватели***, которые могут их вести; Рекомендованные ***Учебники***.

Исходный вариант таблицы:

Преподаватели	Курс	Учебники
Иванов	Сист. упр. БД	Коннолли
Иванов	Сист. упр. БД	Дейт
Иванов	Сист. упр. БД	Гарсиа-Молина
Петров	Сист. упр. БД	Коннолли
Петров	Сист. упр. БД	Дейт
Петров	Сист. упр. БД	Гарсиа-Молина
Сидоров	Сист. упр. БД	Коннолли
Сидоров	Сист. упр. БД	Дейт
Сидоров	Сист. упр. БД	Гарсиа-Молина
Иванов	ОРБД	Коннолли
Иванов	ОРБД	Гарсиа-Молина

После декомпозиции:

Преподаватели	Курс
Иванов	Сист. упр. БД
Петров	Сист. упр. БД
Сидоров	Сист. упр. БД
Иванов	ОРБД

Курс	Учебники
Сист. упр. БД	Коннолли
Сист. упр. БД	Дейт
Сист. упр. БД	Гарсиа-Молина
ОРБД	Коннолли
ОРБД	Гарсиа-Молина

7. Зависимости соединения в реляционных таблицах. 5-я нормальная форма и порядок к ней приведения. Примеры.

Пятая нормальная форма (проекционно-соединительная, 5НФ)

До настоящего момента приведение таблицы к очередной нормальной форме заключалось в её декомпозиции на две таблицы. И эта декомпозиция была «без потерь» (или, если точнее – без потерь и ненужных приобретений). Это означает, что после декомпозиции исходный вариант таблицы мог быть восстановлен с помощью операции соединения, при этом никакие исходные строки не терялись, и не возникало новых (ложных) строк.

Однако существуют таблицы, которые не могут быть декомпозированы на две таблицы без потери данных, т. е. при обратном соединении какие-то данные будут потеряны и/или возникнут новые строки, которых не было в изначальной таблице. Но если декомпозировать такую таблицу не на две, а на три таблицы, то подобной «потери данных» (или, говоря точнее, потери целостности) наблюдаться не будет.

Если это свойство таблицы должно выполняться для любых её допустимых состояний, то говорят, что таблица находится в **зависимости соединения.**

Требование 5НФ: реляционная таблица находится в 5НФ тогда и только тогда, когда она находится в 4НФ и не содержит зависимостей соединения.

Пример.

Имеется некоторое множество ПРОЕКТОВ, в каждом проекте имеется несколько НАПРАВЛЕНИЙ РАБОТЫ (инженерные расчёты, разработка программ, бухгалтерия и пр.) Для участия в проекте по какому-либо направлению потенциальный УЧАСТНИК ПРОЕКТА должен обладать соответствующей компетенцией, официально подтверждённой. Кроме того, потенциальный участник должен быть допущен до участия в соответствующем проекте.

8. Варианты организации файлов данных на вторичных устройствах хранения. Порядок поиска и обновления для каждого из вариантов.

Организация файла определяет распределение данных файла по записям и страницам на вторичном устройстве хранения.

Варианты организации:

1. Последовательная неупорядоченная.
2. Последовательная упорядоченная.
3. Хешированная.

1. Неупорядоченные последовательные файлы.

Записи размещаются в порядке поступления. Каждая новая запись – на последнюю страницу. Если не хватает места, создаётся новая страница.

Доступ: при отсутствии индексов – только линейный поиск.

Удаление записи: загружается страница, удаляется запись, сохраняется, сбрасывается на диск. Освободившееся пространство повторно не используется, что влечёт необходимость периодической физической реорганизации файла со стороны администратора.

2. Упорядоченные последовательные файлы.

Записи в файле хранятся в отсортированном виде по значению ключа упорядочения.

Поиск (по ключу упорядочения): двоичный.

Вставка: осложняется необходимостью поддерживать порядок. Если на нужной странице достаточно места, то потребуется переупорядочить только её. Если нет – обычно используется временный несортированный **файл переполнения**. Его содержимое периодически сливается с основным файлом.

Поиск в файле переполнения – линейный.

При *удалении* записи необходима периодическая реорганизация файла, чтобы ликвидировать пустые места.

3. Хешированные файлы.

Для определения адреса страницы, где находится нужная запись, используется **хеш-функция**, аргументами которой являются значения одного или нескольких **полей перемешивания**. Если поле перемешивания является ключевым, то оно называется **хеш-ключом**. Хеш-функция выбирается так, чтобы записи по её значениям были распределены максимально равномерно. Исходный файл делится, таким образом, на более мелкие **сегменты** (каждый из которых соответствует конкретному значению хеш-функции), и поиск происходит не во всём файле, а в каком-либо сегменте. Сегменты состоят из **слотов** (ячеек), предназначенных для размещения отдельных записей. В пределах одного сегмента записи размещаются в слотах в порядке поступления.

1. *Открытая адресация. С*истема выполняет линейный поиск первого доступного слота для вставки в него новой записи. После неудачного поиска пустого слота в последнем сегменте – поиск продолжается с первого сегмента. При поиске по полю перемешивания запись констатируется отсутствующей при обнаружении первого пустого слота.
2. *Несвязанная область переполнения.* Она используется для размещения записей, которые не могут быть вставлены по вычисленному для них адресу хеширования. В области переполнения производится линейный поиск.
3. *Связанная область* переполнения. Каждому сегменту выделяется дополнительное поле *(указатель синонима)*. Оно определяет наличие конфликта и указывает страницу в области переполнения, использованную для его разрешения. Если указатель равен нулю, то конфликта нет.

4. *Многократное хеширование.* Альтернативный способ разрешения конфликтов заключается в применении второй хеш-функции, если первая функция приводит к возникновению конфликта. Цель второй хеш-функции заключается в получении нового адреса хеширования, который позволил бы избежать конфликта. Вторая хеш-функция обычно используется для размещения записей в области переполнения.

9. Индексы, их устройство и разновидности. Примеры. Организация поиска с участием индексов. Многоуровневые индексы.

Индекс – структура данных, которая помогает СУБД быстрее обнаружить отдельные записи а файле и сократить время выполнения запросов пользователей.

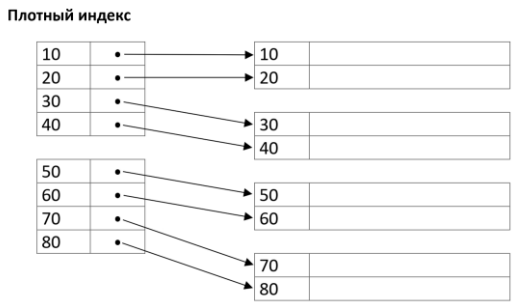
Структура индекса связана с определенным ключом поиска и содержит записи, состоящие из ключевого значения и адреса логической записи в файле, содержащей это ключевое значение. Значения в индексном файле упорядочены по полю индексирования, которое обычно строится на базе одного атрибута.

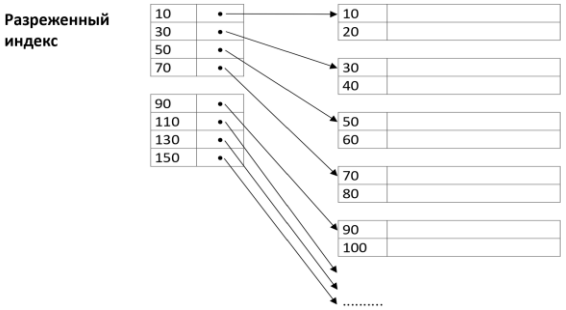
Типы индексов.

1. *Первичный индекс.* Файл данных упорядочен по ключевому полю, и индекс построен на основе значений этого поля.
2. *Индекс кластеризации.* Файл данных упорядочен по неключевому полю, и индекс построен на основе значений этого поля.
3. *Вторичный индекс.* Индекс строится по полю, не являющемуся полем упорядочивания.

Плотный индекс – содержит индексные значения для всех ключевых значений поиска.

Разреженный индекс – содержит индексные значения только для некоторых (опорных) ключевых значений поиска.





Многоуровневые индексы.

При возрастании размера индексного файла и расширении его содержимого на большое количество страниц время поиска нужного индекса также значительно возрастает. Даже если для отыскания элемента индекса используется алгоритм двоичного поиска, системе зачастую приходится выполнять немалое количество операций дискового ввода-вывода. Обратившись к многоуровневому индексу, можно попробовать решить эту проблему путем сокращения диапазона поиска. Данная операция выполняется над индексом посредством расщепления индекса на несколько субиндексов меньшего размера и создания индекса для этих субиндексов.

На первом этапе для ускорения просмотра индексного файла можно создать дополнительный более разреженный индекс второго уровня – «индекс для индекса». С возрастанием уровня индексы должны быть всё более разреженными.

Вторичные (secondary) индексы

Файл данных, связанный со вторичным индексом, необязательно должен быть отсортирован по ключу этого индекса.

И ключ вторичного индекса может содержать повторяющиеся значения.

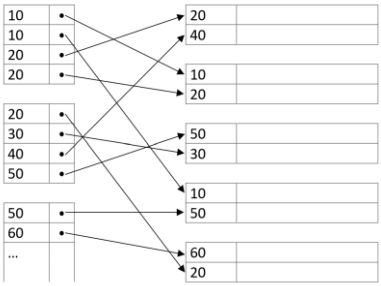
Как правило, вторичный индекс – это плотный индекс, содержащий дубликаты некоторых ключевых значений.

Как и любой индекс, он состоит из пар «ключ-значение», отсортированных по значениям ключа.

Если необходимо дополнить структуру индекса вторым уровнем – второй уровень должен быть разреженным.

Указатели в записях одного блока вторичного индекса (в отличие от первичного) способны адресовать самые разные блоки данных (а не один или несколько последовательных блоков).

Например, для поиска данных с ключом 20 потребуется обратиться к двум блокам индекса и последовать по указателям к трём различным блокам данных



10. B$^+$-деревья и их свойства. Поиск с помощью B$^+$-деревьев. Вставка новых элементов в B$^+$-дерево. Примеры.

B$^+$-деревья

Во многих СУБД для хранения данных и/или индексов используется структура, называемая **деревом**. Дерево представляет иерархию **узлов**, в которой у каждого отдельно взятого узла могут быть: один **родительский** и один или несколько (или ни одного) **дочерних** узлов.

Узлы дерева могут быть разделены на три категории:

- один **корневой** узел: у него нет родительского узла;
- узлы-**листья**: у них нет дочерних узлов;
- узлы промежуточных уровней.

Глубиной дерева называется максимальной количество уровней между корнем и листом. Если она одинакова для всех листов, дерево называется **сбалансированным** или **B-деревом**.

Степенью или **порядком** дерева называется максимально допустимое количество дочерних узлов для каждого родительского узла.

Бинарным деревом называется дерево порядка 2, в котором каждый узел имеет не более 2 дочерних узлов.

Поскольку время доступа в древовидной структуре в большей степени зависит от глубины, чем от ширины, обычно принято использовать более «кустистые» и менее глубокие деревья.

Типичное B^+-дерево содержит 3 уровня: корень, вершины одного промежуточного уровня и листья.

Каждый узел B^+-дерева представляет собой индексный блок, состоящий из n значений индексного ключа и $n+1$ указателей. Величина n определяется объёмом блока.

Вид узла дерева порядка 4:

B^+-дерево определяется следующим образом:

- Дерево должно быть сбалансированным в любой момент времени;
- Корень (если он не лист) должен иметь не менее 2 дочерних узлов;
- В дереве порядка n каждый промежуточный узел должен иметь от $[(n+1)/2]$ до n дочерних узлов;
- В дереве порядка n количество ключевых значений в листе должно находиться в пределах от $[n/2]$ до $n–1$;
- Количество ключевых значений в нелистовом узле на единицу меньше количества указателей;
- Листы дерева связаны указателями в порядке возрастания ключевых значений.

На практике каждый узел в B^+-дереве является страницей (дисковым блоком, обрабатываемым одной операцией ввода-вывода).

Пример. Пусть дисковый блок имеет размер 4096 байт, ключи представляют 4-байтные целочисленные значения, под каждый указатель отводится также 4 байта, и каждый блок содержит 4-байтный указатель на следующий узел того же уровня. В таком случае в одном блоке можно сохранить (4096–4)/(4+4)=511 индексных записей. Таким образом порядок данного B^+-дерева равен 512. Корень может содержать до 511 записей и иметь до 512 дочерних узлов. Каждый дочерний узел может также содержать до 511 записей, что в целом даёт структуру из 261 632 записей. В свою очередь каждый дочерний узел тоже может иметь до 512 дочерних узлов, что в сумме даёт 262 144 дочерних узла на 2-м уровне дерева. Каждый из этих узлов также может содержать до 511 записей, что даёт двухуровневую структуру из 133 955 584 записей. Если все уровни просуммировать, то теоретический максимум для количества индексных записей в двухуровневом дереве составит 134 217 727.

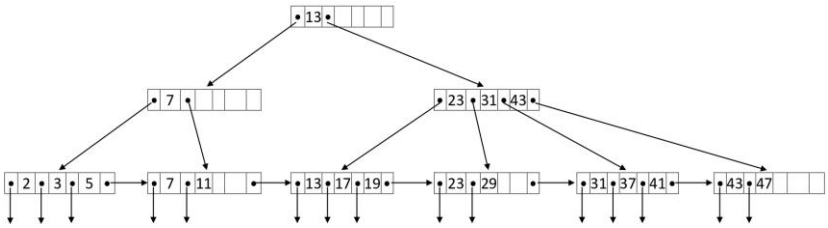
Таким образом, описанная структура B^+-дерева позволяет осуществить произвольный доступ к отдельной записи таблицы, содержащей 134 217 727 записей, с выполнением лишь 4 операций доступа к диску.

Вставка новых элементов в B^+-дерево

Процесс рекурсивен:

- Найти в соответствующем блоке-листе свободное место подходящего объёма и вставить новую пару «ключ-указатель».
- Если свободное место в листе отсутствует, расщепить лист на 2 листа и распределить между ними множество «ключей-указателей» поровну (или примерно поровну).
- Это расщепление влечёт необходимость вставки новой пары «ключ-указатель» в соответствующую вершину следующего более высокого уровня. Применяется та же стратегия.
- Особая ситуация: при попытке вставки «ключ-указатель» в корневую вершину обнаруживается, что свободного места нет. В этом случае корневая вершина расщепляется на две вершины и создаётся новая корневая вершина.

Пример. Вставка в B^+-дерево нового ключа 40.



Модификации подлежит 5-й слева блок-лист (2-й справа). Но в нём нет свободного места. Необходимо расщеплять.

11. Градации алгоритмов выполнения реляционных операций и классы этих операций. Технология выполнения однопроходного алгоритма для унарной операции над отдельными кортежами.

1. **Табличное сканирование.** Кортежи таблицы компактным образом размещаются в определённой группе блоков вторичной (дисковой) памяти, и адреса блоков известны. В этом случае система способна загружать блоки один за другим.
2. **Индексное сканирование.** Местоположение блоков может быть неизвестно, но для некоторого атрибута сконструирован индекс. Этот индекс (даже если он разреженный) может привести ко всем блокам, содержащим данные таблицы.

По использованию дополнительных структур и механизмов:

1. Алгоритмы без использования дополнительных механизмов.
2. Алгоритмы, основанные на сортировке.
3. Алгоритмы, основанные на хешировании.
4. Алгоритмы, основанные на индексировании.

1. **Однопроходные алгоритмы.** Данные считываются с диска только один раз. Используются, когда хотя бы одна из таблиц-операндов может быть целиком загружена в оперативную память.
2. **Двухпроходные алгоритмы.** Относятся к случаю, когда объём данных превышает размер свободной области оперативной памяти. Предусматривается необходимость считывания данных с диска, манипуляции с ними в памяти, сохранения всей (или почти всей) информации обратно на диске, с последующей повторной загрузкой с целью обработки в процессе второго прохода.
3. **Многопроходные алгоритмы.** Используются, когда данные имеют особо крупный объём. Алгоритм представляет собой рекурсивное обобщение двухпроходной технологии.

Однопроходные алгоритмы

Операторы будем разделять на 3 класса:

1. *Унарные операции над отдельными кортежами:* селекция σ, проекция π. Они не требуют наличия в оперативной памяти содержимого всей таблицы и предполагают считывание с диска отдельных блоков данных, использование одного буфера оперативной памяти и последовательного вывода результатов.
2. *Унарные операции над полными таблицами:* удаление дубликатов δ, группировка γ. Предусматривают загрузку всех или большей части кортежей в оперативную память, поэтому однопроходные алгоритмы способны работать только с такими таблицами, содержимое которых может быть сохранено в *M* доступных буферах оперативной памяти.
3. *Бинарные операции над полными таблицами:* объединение, пересечение, разность, декартово произведение, все соединения. Требуется, чтобы размер хотя бы одного операнда был ограничен величиной *M*.

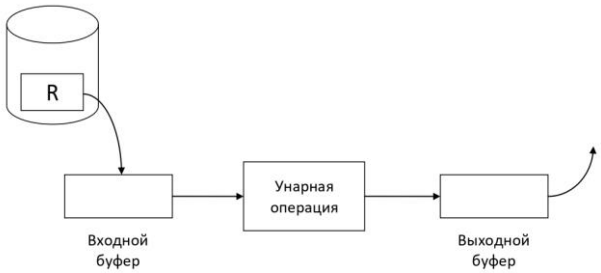
12. Технология однопроходного алгоритма для выполнения операции удаления кортежей-дублей.

Унарные операции над отдельными кортежами

Очередной блок считывается с диска в оперативную память, выполняется операция над каждым кортежем загруженной порции, далее выбранные (или спроецированные) кортежи перемещаются в выходной буфер.

Если производится селекция по условию сопоставления некоторого атрибута с определённой константой, и таблица проиндексирована по этому атрибуту, процесс существенно упрощается.

Поскольку в этом случае перебираются не все блоки, а только некоторые.



Унарные операции над полными таблицами.

Удаление кортежей-дубликатов

Для удаления дубликатов блоки с данными таблицы R могут считываться последовательно, но при загрузке очередного кортежа следует принять одно из решений:

- Если кортеж ранее не встречался, его надлежит скопировать в выходной буфер;
- Если аналогичный кортеж прежде уже загружался, его следует игнорировать.

Чтобы реализовать эти требования, достаточно обеспечить хранение в оперативной памяти каждого уникального считанного кортежа. Один (входной) буфер используется для хранения «текущего» загруженного блока таблицы R, оставшиеся *M*–1 буферов – для хранения копий всех ранее просмотренных уникальных кортежей.

13. Варианты алгоритмов выполнения операции соединения. Описание группирования операндов соединений с помощью деревьев. Разновидности деревьев.

Группирование

В операторе группирования задаются два списка атрибутов: поля группировки и поля-аргументы агрегативных функций. В оперативной памяти должно создаваться по одному *элементу* для каждого совокупного значения полей группировки. Это означает, что кортежи таблицы R можно сканировать последовательно, по одному блоку. Конечный выходной *элемент* для каждой группы состоит из значений полей группировки и значений соответствующих функций для этой группы.

Пусть a – некоторый атрибут таблицы R. Тогда алгоритмы накопления значений агрегативных функций стандартные:

- Для функций Min(a) и Max(a): сравнить значение атрибута a загруженного кортежа с хранимым минимальным (максимальным) значением; если значение атрибута a загруженного кортежа меньше (больше) хранимого минимального (максимального) значения, то сохранить его в виде минимального (максимального) значения.
- Для функции Count: увеличить значение счётчика на единицу для каждого кортежа, относящегося к рассматриваемой группе.
- Для функции Sum(a): прибавить значение атрибута a загруженного кортежа к накапливаемой сумме значений a группы.
- Для функции Avg(a) требуется поддерживать два аккумулируемых значения: счётчик количества кортежей в группе и сумму значений атрибутов a этих кортежей. После того, как все кортежи загружены, следует разделить итоговую сумму на итоговое количество.

В любом случае до тех пор, пока не будет обработан последний кортеж, итог выполнения оператора группировки (группировки-агрегирования) получить нельзя.

Для повышения эффективности обработки кортежей в оперативной памяти можно использовать структуру данных, позволяющую находить элемент для каждой группы по заданным значениям группирующих атрибутов. Для этой цели подойдут общеупотребительные структуры данных, ориентированные на обработку информации в

оперативной памяти – такие как хеш-таблицы или сбалансированные деревья поиска. Однако роль ключа поиска могут выполнять только группирующие атрибуты.

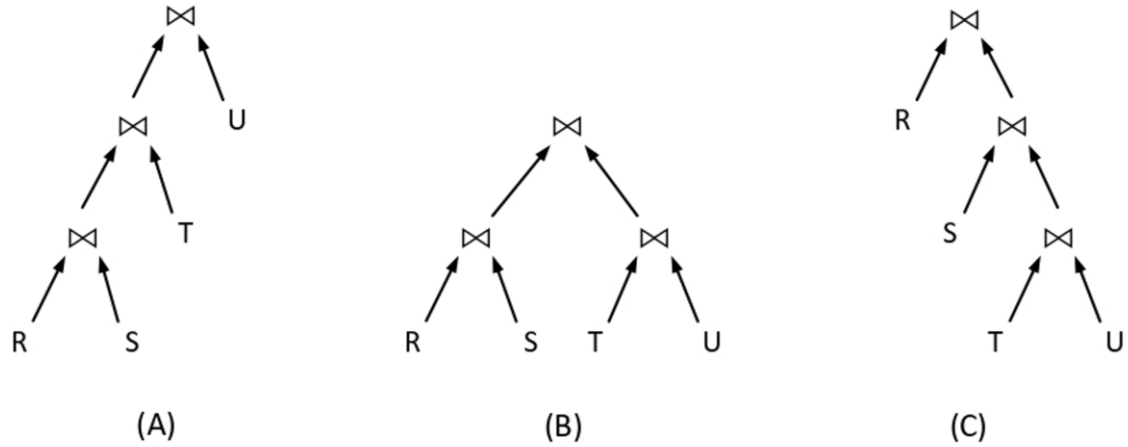
Бинарные операции. Соединение

Операция соединения хоть и является теоретически симметричной и перестановочной, но конкретные алгоритмы её исполнения используют её операнды (левый и правый) в принципиально разных ролях. Среди алгоритмов реализации соединения наиболее популярны следующие:

- соединение блоками с использованием вложенных циклов;
- соединение блоками с использованием вложенных циклов и индексов;
- соединение посредством сортировки-слияния;
- хешированное соединение.

Так, при соединении **с использованием вложенных циклов** левый операнд (так называемый **опорный**, и предпочтительно, чтобы он был меньшего объёма, чем правый) целиком считывается в буфер оперативной памяти, а правый операнд (**тестируемый**) считывается в буфер отдельными блоками. Далее во внешнем цикле выполняется последовательный перебор строк опорного операнда, а во внутреннем цикле последовательно просматриваются строки блока тестируемого операнда. И в случае выполнения соединительного условия две строки сцепляются и помещаются в буфер для результирующей таблицы. Если тестируемый операнд оказывается проиндексированным по полю соединения, то просмотр всего блока тестируемого операнда заменяется более эффективным поиском по индексу.

Порядок соединений удобно изображать в виде дерева, на котором сразу выделяются их левые и правые операнды.



Соединение с помощью сортировки-слияния

Для соединений по эквивалентности наиболее эффективный алгоритм реализуется в том случае, если обе таблицы отсортированы по соединяемым атрибутам. В этом случае можно отыскать подходящие строки таблиц R и S, выполнив **слияние** двух таблиц. (Если исходные таблицы не отсортированы, можно предварительно выполнить их сортировку). Поскольку строки таблиц отсортированы, те из них, которые содержат одинаковые значения соединяемого атрибута, гарантированно будут размещаться рядом. Если

предположить, что соединение выполняется для связи типа «многие ко многим» (т. е. в обоих таблицах может существовать несколько строк с одним и тем же соединяемым значением), а также принять, что любой набор строк с одним и тем же соединяемым значением может быть полностью помещен в буфер базы данных, то каждый блок каждой из соединяемых таблиц достаточно будет считать только один раз.

Соединение с помощью хеширования

Алгоритм соединения с помощью хеширования может использоваться в случае естественного соединения (или соединения по эквивалентности) таблиц R и S по атрибуту A. Суть этого алгоритма состоит в следующем: на первом этапе, который называется **этапом хеширования**, происходит разбиение таблиц R и S на разделы в соответствии со значениями некоторой хеш-функция, результаты вычисления которой дают равномерно распределенные случайные значения. Каждый полученный эквивалентный раздел таблиц R и S должен иметь одни и те же значения соединяемых атрибутов, хотя он может содержать больше одного их значения. Алгоритм предусматривает проверку эквивалентных разделов на наличие одного и того же значения.

Например, если таблица R с помощью некоторой хеш-функции h(…) разбито на разделы R_1, R_2, …, R_m , а таблица S — на разделы S_1, S_2, …, S_m , то если B и C являются атрибутами таблиц R и S соответственно, а $h(R.B)\neq h(S.C)$, то $R.B\neq S.C$. Но если $h(R.B)=h(S.C)$, это не обязательно означает, что $R.B=S.C$, поскольку одно и то же значение хеш-функции может быть получено для разных исходных значений.

На втором этапе, который называется **этапом проверки**, последовательно считывается каждый из разделов R и предпринимается попытка соединить содержащиеся в нем строки со строками эквивалентного раздела S. Если второй этап выполняется с помощью соединения вложенными циклами, то во внешнем цикле применяется раздел меньшего размера, допустим R_i. Весь раздел R_i считывается в память, после чего с диска считывается каждый блок эквивалентного ему раздела S_i и в нём каждая строка используется для проверки наличия соответствующих строк в разделе R_i.

Для повышения эффективности обычно в оперативной памяти формируется хеш-таблица для каждого раздела R_i с применением второй хеш-функции, отличной от той, с помощью которой выполняется разбиение таблиц на разделы.

14. Двухпроходные и многопроходные алгоритмы выполнения реляционных операций. Пример двухпроходного алгоритма для удаления кортежей-дублей.

Двухпроходные алгоритмы

Многопроходные алгоритмы применяются в условиях, когда таблицы-операнды настолько велики, что не могут быть целиком загружены в оперативную память и, соответственно, не могут быть обработаны однопроходными алгоритмами.

Двухпроходные алгоритмы предусматривают загрузку содержимого таблиц-аргументов в оперативную память, некоторые действия по обработке их в оперативной памяти, затем сохранение на диске и повторное считывание с действиями по завершению операции.

Эта схема может быть распространена на произвольное количество проходов, при которых данные сохраняются на диске и загружаются в оперативную память многократно.

Однако даже для обработки весьма крупных таблиц обычно бывает достаточно двух проходов.

Механизм сортировки (вспомогательный)

Если размер B(R) таблицы R превышает количество *M* доступных буферов оперативной памяти, надлежит **циклически** выполнять действия:

1. Загрузить очередные *M* блоков таблицы R в оперативную память.
2. Упорядочить содержимое считанных блоков в памяти, используя один из эффективных алгоритмов сортировки.
3. Сохранить отсортированный подсписок в *M* дисковых блоках.

По завершении первого прохода предусматривается проведение второго прохода, со «слиянием» отсортированных подсписков в порядке, соответствующем природе оператора.

Удаление кортежей-дубликатов (в два прохода)

1-й проход. Кортежи R распределяются по отсортированным подспискам и сохраняются в дисковых блоках.

2-й проход. Все доступные буферы оперативной памяти используются для загрузки по одному блоку из каждого подсписка. Но вместо сортировки каждый *уникальный* кортеж копируется в результирующую таблицу, а другие идентичные кортежи (дубли) игнорируются.

Иными словами:

Просматриваются первые кортежи каждого блока, среди них отыскивается первый в порядке сортировки (назовём его *t*). Одна копия *t* направляется в результирующую таблицу, остальные удаляются из блоков подсписков. Если в этот момент некоторый блок исчерпывается, в буфер считывается очередной блок того же подсписка. Если во вновь загруженном блоке существуют экземпляры *t*, они тоже удаляются.

15. Системы OLTP и хранилища данных. Свойства хранилищ и их сравнение со свойствами систем OLTP.

Сравнение систем OLTP и хранилищ данных

СУБД, созданная для поддержки оперативной обработки транзакций (OLTP), обычно рассматривается как непригодная для организации хранилищ данных, поскольку к этим двум типам систем предъявляются совершенно разные требования.

Например: системы OLTP проектируются с целью обеспечения максимально интенсивной обработки **фиксированных транзакций**, тогда как хранилища данных — прежде всего для обработки единичных **произвольных запросов**.

Организация обычно имеет **несколько различных систем OLTP**, предназначенных для поддержки различных деловых процессов (управление запасами, выставление счетов клиентам, продажа товаров и пр.). Эти системы вырабатывают **оперативные данные**, которые являются очень подробными, текущими и подверженными изменениям. Системы

OLTP оптимально подходят для интенсивной обработки транзакций, которые **проектируются заранее**, многократно повторяются и связаны преимущественно с обновлением данных. В соответствии с этими особенностями, данные в системах OLTP организованы согласно требованиям конкретных деловых приложений и позволяют **принимать повседневные решения** большому количеству параллельно работающих пользователей-исполнителей.

И в организации обычно имеется **только одно хранилище данных**, которое содержит исторические, подробные, до определённой степени обобщённые и практически неизменные данные (т. е. новые данные могут только добавляться). Хранилища данных предназначены для обработки **относительно небольшого количества транзакций**, которые имеют непредсказуемый характер и требуют ответа на **произвольные, неструктурированные и эвристические запросы**. Информация в хранилище данных организована в соответствии с требованиями возможных запросов и предназначена для поддержки принятия долговременных стратегических решений относительно небольшим количеством руководящих работников.

Хотя системы OLTP и хранилища данных имеют совершенно разные характеристики и создаются для различных целей, все же они тесно связаны в том смысле, что системы OLTP являются источником информации для хранилища данных.

Основная проблема при организации этой связи заключается в том, что поступающие из систем OLTP данные могут быть, **несогласованными, фрагментированными, подверженными изменениям, содержащими дубликаты или пропуски**. Поэтому до размещения в хранилище эти оперативные данные должны быть «очищены».

Системы OLTP не предназначены для получения быстрого ответа на произвольные запросы. Они также не используются для хранения устаревших исторических данных, которые требуются для анализа тенденций. Системы OLTP в основном поставляют огромное количество необработанных данных, которые анализу поддаются непросто.

16. Основные проблемы разработки и сопровождения хранилищ данных.

Основные проблемы разработки и сопровождения хранилищ данных

1. Недооценка ресурсов, необходимых для загрузки данных;

2. Скрытые проблемы источников данных;

3. Отсутствие требуемых данных в имеющихся архивах;

4. Повышение требований конечных пользователей;

5. Унификация данных;

6. Высокие требования к ресурсам;

7. Владение данными;

8. Сложное сопровождение;

9. Долговременный характер проектов;

10. Сложности интеграции.

17. Архитектура хранилищ данных. Потоки данных в хранилищах.

Архитектура хранилищ данных

Основные компоненты:

1. Оперативные данные.
2. Менеджер загрузки.
3. Менеджер хранилища.
4. Менеджер запросов.
5. Детальные данные.
6. Частично обобщённые данные.
7. Глубоко обобщённые данные.
8. Архивные и резервные копии.
9. Метаданные.
10. Средства доступа к данным конечных пользователей.

18. Требования к СУБД для хранилища данных и их обоснование.

Требования к СУБД для хранилища данных

1. Высокая производительность загрузки данных.
2. Возможность обработки данных во время загрузки.
3. Наличие средств управления качеством данных.
4. Высокая производительность запросов.
5. Масштабируемость по объёму.
6. Масштабируемость по количеству пользователей.
7. Возможность организации сети хранилищ данных.
8. Наличие средств администрирования хранилища.
9. Поддержка интегрированного многомерного анализа размерностей.
10. Расширенный набор функциональных средств запросов.

Основная проблема – большой объём базы данных хранилища.

При работе с подобной базой данных становится особенно важным возможность обеспечения **параллельной работы**.

Параллельные СУБД могут одновременно выполнять сразу несколько операций с базой данных, разбивая отдельные задачи на малые части таким образом, чтобы они могли быть распределены между несколькими процессорами.

В частности, большие и сложные запросы система должна разбивать на более мелкие подзапросы, параллельно запускать эти мелкие подзапросы на выполнение, а затем собирать вместе полученные результаты.

Существуют две основные архитектуры аппаратного обеспечения для выполнения параллельных вычислений, которые могут использоваться в качестве платформы для сервера базы данных в хранилищах данных:

- Симметричная мультипроцессорная обработка (Symmetric MultiProcessing — SMP). Группа тесно связанных процессоров, которые совместно используют оперативную и дисковую память.
- Массовая мультипроцессорная обработка (Massively Multi-Processing — MMP). Группа слабо связанных процессоров, каждый из которых использует свою собственную оперативную и дисковую память.

19. Метаданные в хранилищах данных, их содержание и функции.

Метаданные (данные о данных)

Метаданные — это описание информационного содержания хранилища данных: что в нём содержится, откуда что поступает, какие операции выполнялись во время очистки, как осуществлялись интеграция и обобщение данных.

Метаданные используются в самых разных целях, и управление метаданными является важнейшим вопросом при создании полностью интегрированного хранилища данных.

Одним из основных назначением метаданных является сохранение информации о происхождении данных, чтобы администраторы хранилища могли знать историю любого элемента данных, помещенного в хранилище.

Однако в пределах хранилища данных метаданные выполняют несколько функций, связанных с преобразованием и загрузкой данных, обслуживанием хранилища данных и формированием запросов.

Структура метаданных для разных процессов может различаться в зависимости от их назначения. Это означает, что для одного и того же элемента данных в хранилище может храниться сразу несколько вариантов метаданных. Кроме того, большинство компаний-разработчиков применяет собственные версии структуры метаданных в своих инструментах управления копированием данных и средствах доступа, предназначенных для конечных пользователей.

В частности, инструменты управления копированием данных используют метаданные для определения правил отображения, которые необходимо применить для преобразования исходных данных в общепринятую форму.

Средства доступа конечных пользователей к данным используют метаданные для выбора способа построения запроса. В целом управление метаданными внутри хранилища данных является очень сложной задачей.

Метаданные, связанные с преобразованием и загрузкой данных, должны описывать источник данных и любые изменения, внесенные в эти данные. Например, для каждого исходного поля должны храниться такие сведения, как уникальный идентификатор, исходное имя поля, тип источника данных, исходное расположение, включая системное и объектное имя, а также тип преобразованных данных и имя таблицы назначения. Если поле подвергается каким-то изменениям, начиная с простого изменения его типа и вплоть до выполнения над ним сложного набора процедур и функций, то все эти изменения также должны быть зафиксированы.

Метаданные, связанные с управлением данными, описывают способ хранения данных в хранилище. Должен быть описан каждый объект базы данных, включая данные,

содержащиеся во всех таблицах, индексах и представлениях. Кроме того, должны быть описаны и любые связанные с этими объектами ограничения.

Эта информация хранится в системном каталоге СУБД с учетом некоторых дополнительных ограничений, присущих хранилищам данных. Например, метаданные должны описывать любые поля, связанные с агрегированными данными, включая описание способа агрегирования. Кроме того, должно быть описано секционирование таблиц, в том числе ключ секционирования, а также диапазон данных, связанных с каждой секцией.

Описанные выше метаданные требуются также диспетчеру запросов для формирования соответствующих запросов. Диспетчер запросов, в свою очередь, вырабатывает дополнительные метаданные о выполняемых запросах, которые могут быть использованы для накопления исторических данных обо всех выполненных запросах и формирования профилей запросов для каждого пользователя, группы пользователей или хранилища данных в целом. Существуют также метаданные, связанные с пользователями запросов, которые включают, например, информацию о том, что подразумевается под терминами "цена" или "клиент" в конкретной базе данных и изменялся ли смысл этих терминов со временем.

Важнейшим вопросом интеграции является **синхронизация метаданных** разного типа, используемых в пределах всего хранилища данных. Поскольку разные инструменты хранилища данных создают и используют свои собственные метаданные, для достижения полной интеграции необходимо добиться того, чтобы эти инструменты могли пользоваться метаданными совместно. Задача в этом случае заключается в синхронизации метаданных между разными программными продуктами компаний-разработчиков, использующих несовместимые хранилища метаданных. Например, необходимо определить нужный элемент метаданных на соответствующем уровне детализации одного программного продукта, потом установить его соответствие другому элементу метаданных на соответствующем уровне детализации другого программного продукта, а затем отрегулировать любые существующие между ними различия в способах представления.

Эту операцию следует повторить для всех других элементов метаданных, общих для двух программных продуктов.

Более того, любые изменения метаданных или даже «метаметаданных» в одном программном продукте должны быть переданы другому программному продукту. Задача синхронизации двух программных продуктов очень сложна, поэтому повторение такого процесса для шести или более продуктов, которые образуют программное обеспечение хранилища данных, может потребовать много ресурсов. Тем не менее синхронизация метаданных обязательно должна быть выполнена.

Решить данную проблему можно двумя способами:

- с помощью механизмов автоматической передачи метаданных между хранилищами метаданных разных инструментов;
- путем создания репозитария метаданных.

20. Магазины (киоски, витрины) данных. Причины их создания, их основные характеристики и отличие от хранилищ. Архитектура магазинов данных.

Магазины (киоски, витрины) данных (Data Mart)

Магазин (киоск, витрина) данных:

Подмножество хранилища данных, представляющее собой массив тематической, узконаправленной информации, которое поддерживает требования отдельного подразделения или деловой сферы организации.

Это подмножество данных хранилища обычно представлено в виде **обобщённой информации**, связанной с некоторым подразделением или деловой сферой предприятия. Магазин данных может быть независимым или определённым образом связанным с централизованным хранилищем данных.

Основные различия между магазином и хранилищем данных.

• Магазин данных отвечает требованиям пользователей только одного из подразделений организации или некоторой ее деловой сферы;

• Магазин данных обычно не содержит подробных оперативных сведений (в отличие от хранилища данных);

• Поскольку магазин данных содержит меньше информации, чем хранилище, структура информации магазина данных более понятна и проста в управлении.

Для магазина данных можно выбрать двух- или трёхуровневую архитектуру.

Данные могут быть распределены между этими тремя уровнями.

Причины создания магазинов данных

1. Для предоставления пользователям доступа к данным, которые приходится анализировать чаще других.
2. Для предоставления данных группе пользователей некоторого отдела или деловой сферы в форме, которая соответствует их коллективному представлению о данных.
3. Для сокращения времени ответа на запрос (за счет сокращения объёма обрабатываемых данных).
4. Для предоставления данных, структурированных в соответствии с требованиями доступа к данным. Для многих инструментов доступа к данным, таких как инструменты разработки данных или оперативной аналитической обработки (OLAP), может потребоваться создать отдельную внутреннюю структуру базы данных. (На практике подобные инструменты часто создают свои собственные магазины данных, предназначенные исключительно для поддержки их специфических функций).
5. Магазины данных обычно содержат меньший объём данных, поэтому такие задачи, как очистка, загрузка, преобразование и интеграция данных, выполняются проще. Следовательно, реализация и настройка магазина данных требует меньше усилий, чем разработка и реализация корпоративного хранилища данных.
6. Стоимость реализации магазина данных обычно существенно ниже, чем стоимость создания хранилища данных.
7. Круг потенциальных пользователей магазина данных более четко определён, поэтому учесть их требования и организовать необходимую поддержку проще, чем в случае корпоративного хранилища данных.

Характеристики магазинов данных

1. Функциональность.
2. Размер базы данных.
3. Производительность загрузки данных.
4. Доступ пользователей к данным в нескольких магазинах данных.
5. Доступ к магазинам данных через Internet/внутреннюю сеть.
6. Администрирование.
7. Установка.

21. Проектирование хранилищ данных. Понятие о DM-моделировании. Схемы «звезда», «снежинка» и «звезда-снежинка». Денормализация реляционных таблиц и её обоснование. Преимущества DM-моделирования.

При проектировании базы данных для хранилища или магазина данных необходимо иметь представление о том, как эти данные будут использоваться.

БД должна быть спроектирована таким образом, чтобы произвольные запросы пользователей выполнялись с приемлемой производительностью.

В хранилище данных большое количество запросов будет относиться к **детальным** данным, которые могут анализироваться самыми разными способами.

При проектировании БД хранилищ практикуется метод **моделирования размерностей** (DM – Dimensional Model).

В моделировании размерностей используются концепции ER-моделирования с некоторыми важными ограничениями. Каждая модель DM состоит из таблицы с составным первичным ключом, называемой **таблицей фактов**, и набора небольших таблиц — **таблиц размерностей**. Каждая таблица размерностей имеет простой (несоставной) первичный ключ, который точно соответствует одному из компонентов составного ключа в таблице фактов.

Иными словами, первичный ключ таблицы фактов состоит из нескольких внешних ключей.

Эта централизованная структура называется **схемой «звезда»** или **звездообразным соединением**.

Схема «звезда»:

Логическая структура, в центре которой находится **таблица фактов**, содержащая фактические данные, и окружённая содержащими ссылочные данные **таблицами размерностей** (которые могут быть денормализованы).

Ещё одна особенность модели DM: все естественные ключи заменены **искусственными ключами**. Каждый искусственный ключ должен иметь обобщённую структуру, основанную на использовании натуральных чисел. Например, каждое отделение компании имеет естественный ключ ОтделNo и искусственный ключ ОтделID. Применение

искусственных ключей позволяет данным хранилища иметь некоторую независимость от данных.

В схеме «звезда» используются характеристики фактических данных, такие как **факты**, зафиксированные с учётом событий, уже произошедших в прошлом. И модификация этих фактических данных маловероятна.

Поскольку большая часть данных в хранилище данных представлена в виде фактов, таблицы фактов могут иметь огромный размер до сравнению с таблицами размерностей. Поэтому факты рассматриваются как справочные данные, доступные **только для чтения** и **не изменяющиеся** с течением времени.

Часто таблицы фактов содержат некоторое количество неключевых атрибутов (обычно числовых), связанных с каждой записью. Например (на схеме): Исходная цена, Продажная цена, Комиссия.

Атрибуты таблиц размерностей, наоборот, обычно содержат описательную текстовую информацию. Атрибуты размерностей используются в качестве ограничений в запросах к хранилищу данных.

Например, с помощью атрибута «Город» могут поддерживаться запросы, требующие доступа к данным о продаже объектов недвижимости в Москве.

Схема «звезда» может использоваться для ускорения выполнения запросов путём **денормализации** справочной информации, с образованием единой таблицы размерностей.

Например: несколько таблиц размерностей (ОбъектНедвиж, Клиент, Владелец, Сотрудник, Отделение) содержат данные о местонахождении (Город, Район, Регион), которые повторяются в каждой таблице (нарушается 3-я нормальная форма).

Денормализация целесообразна, если имеется некоторое количество объектов, связанных с таблицей размерностей, к которым часто осуществляется доступ.

В этом случае денормализация позволяет избежать накладных расходов, связанных с присоединением дополнительных таблиц для доступа к этим атрибутам.

Однако денормализацию не следует применять в тех случаях, если дополнительные данные требуются не очень часто, поскольку накладные расходы, связанные с просмотром расширенной таблицы размерностей, могут свести на нет любой выигрыш в повышении скорости выполнения запроса.

Схема «снежинка»:

Вариант схемы «звезда», в которой таблицы размерностей не содержат денормализованных данных и могут иметь свои собственные размерности.

Это нормализованная таблица размерностей **Отделение** схемы данных о продаже объектов недвижимости. Эти данные о местонахождении (**Город** и **Район**) можно использовать совместно с таблицами размерностей **ОбъектНедвиж**, **Сторудник**, **Клиент**, **Владелец**.

Схема «звезда-снежинка»:

Гибридная структура, которая состоит из комбинации схем «звезда» и «снежинка».

Некоторые размерности могут быть представлены в обеих формах для удовлетворения потребностей запросов разных типов.

Ключом к пониманию связи между DM- и ER-моделями является то, что единая ER-модель обычно разбивается на несколько DM-моделей. Далее эта совокупность объединяется с помощью согласованных (совместно используемых) таблиц размерностей.

Преимущества DM-моделирования:

1. Эффективность. Единообразие структуры, лежащей в основе базы данных, обеспечивает более эффективный доступ к данным различными средствами, включая генераторы отчетов и инструменты для формирования запросов.
2. Возможность удовлетворения изменяющихся требований. Схема «звезда» может адаптироваться к изменениям пользовательских требований, поскольку все размерности в равной степени обеспечивают доступ к таблице фактов.
3. Расширяемость. Типичные изменения, которые поддерживает модель DM, включают в себя: а) добавление новых фактов (такой же степени детализации, как и существующая таблица фактов); б) добавление новых размерностей; в) добавление новых атрибутов размерностей; г) разбиение существующих записей размерностей на записи с меньшим уровнем детализации.
4. Возможность моделировать обычные деловые ситуации. Количество стандартных методов учета типичных моделируемых ситуаций в мире бизнеса постоянно возрастает. Для каждой из этих ситуаций имеется хорошо изученный набор альтернатив, который может быть запрограммирован специальным образом в генераторах отчётов, инструментах создания запросов и в других интерфейсах пользователя.
5. Обработка запросов с предсказуемыми результатами. Приложение хранилища данных, которое обеспечивает поиск на более глубоком уровне, предусматривает просто введение дополнительных атрибутов размерностей из числа атрибутов, содержащихся в единой схеме «звезда». Приложение хранилища данных, которое выполняет поиск на том же уровне, предусматривает связывание отдельных таблиц фактов с помощью совместно используемых (согласованных) размерностей.

22. Понятие о технологии OLAP и обоснование её необходимости при выполнении запросов аналитического плана. Многомерность в OLAP-приложениях и её уровни.

OLAP:

Динамический синтез, анализ и обобщение больших объёмов **многомерных** данных.

Это технология обработки данных, в которой применяется **многомерное представление агрегированных данных** для обеспечения быстрого доступа к стратегически важной информации в целях углублённого анализа.

Эта технология позволяет пользователям просматривать корпоративные данные таким образом, чтобы они могли наилучшим образом изучить истинное положение своего предприятия.

Инструменты БД общего назначения, такие как средства формирования отчетов и обработки запросов, позволяют легко ответить на вопросы типа «кто» и «что», касающиеся прошлых событий. Например, запрос «Какова была общая прибыль по московскому региону за третий квартал 2022 года?» может быть направлен непосредственно в хранилище данных.

Системы OLAP обладают более мощными возможностями. Они способны отвечать и на вопросы типа «что будет, если» и «почему». И поддерживают процесс принятия решений по выбору стратегии дальнейших действий. Типичные расчеты в системе OLAP могут быть намного сложнее по сравнению с обычным агрегированием данных. Например, с их помощью можно найти ответ на такой сложный вопрос: «Как может отразиться на сбыте объектов недвижимости стоимостью свыше 20 миллионов в различных регионах нашей страны повышение затрат на юридическое оформление сделок на 3,5% и снижение государственных налогов на 1,5%?»

Примеры приложений OLAP:

1. **Финансы:** Составление сметы, исчисление себестоимости по объёму хозяйственной деятельности, анализ финансового состояния и финансовое моделирование.
2. **Сбыт:** Анализ и прогнозирование сбыта.
3. **Маркетинг:** Изучение конъюнктуры рынка, прогнозирование сбыта, анализ эффективности рекламы, анализ клиентуры и сегментация рынка/клиентуры.
4. **Производство:** Планирование производства и анализ причин выпуска некачественной продукции.

В технологии OLAP для хранения данных и связей между ними используются **многомерные структуры**, которые удобнее всего представлять как **кубы данных**, состоящие, в свою очередь, из других кубов данных. Каждая сторона куба является размерностью.

Многомерные базы данных очень компактны и обеспечивают простые средства просмотра и манипулирования элементами данных, обладающих многими взаимосвязями. Куб легко может быть расширен с целью включения новой размерности, например, определяющей количество сотрудников компании в каждом городе. Над данными в кубе могут выполняться операции алгебры матриц, что позволяет легко вычислить агрегирующие величины, например, значение среднего дохода на одного сотрудника компании.

Время обработки многомерного запроса зависит от количества ячеек, которые должны быть обработаны динамически. С ростом числа размерностей количество ячеек в кубе данных возрастает экспоненциально. Однако для большинства многомерных запросов требуются лишь **агрегированные данные** высокого уровня.

Следовательно, для создания эффективной многомерной базы данных необходимо предварительно рассчитать (и накопить) все логические промежуточные и окончательные итоговые значения, причём по всем размерностям. Такое предварительное агрегирование может оказаться особенно ценным, если типичные размерности имеют **иерархическую структуру**. Например, размерность времени может иерархически подразделяться на годы, кварталы, месяцы, недели и дни, а размерность географического расположения — на отделения компании, города, районы и регионы.

Многомерность в OLAP-приложениях представляют в виде трех уровней:

1. ***Многомерное представление данных*** – средства конечного пользователя, обеспечивающие многомерную визуализацию и манипулирование данными; слой многомерного представления отделён от физической структуры данных и воспринимает данные как многомерные.
2. ***Многомерная обработка*** – средство (язык) формулирования многомерных запросов (традиционный реляционный язык SQL здесь непригоден) и процессор, умеющий обработать и выполнить такой запрос.
3. ***Многомерное хранение*** – средства физической организации данных, обеспечивающие эффективное выполнение многомерных запросов.

Во всех OLAP-системах первые два уровня присутствуют обязательно.

23. Преимущества систем OLAP по сравнению с системами OLTP. Основные аналитические операции серверов многомерных баз данных на основе OLAP. 12 правил Кодда для систем OLAP.

- Сохранение контроля над целостностью корпоративных данных в масштабах всей организации, поскольку приложения OLAP опираются на хранилища данных и системы OLTP, постоянно получая от них актуальные исходные данные.
- Уменьшение нагрузки на системы OLTP или хранилища данных, связанной с выполнением запросов, а также сокращение сетевого трафика.
- Расширение возможностей получения доходов и повышения прибыльности благодаря быстрому реагированию на потребности рынка.

Серверы многомерных баз данных на основе OLAP могут выполнять перечисленные ниже основные аналитические операции:

- **Консолидация.** Включает такие агрегирующие операции, как простое суммирование значений (свёртка) или расчёт с использованием сложных выражений, включающих промежуточные данные. Например, показатели для отделений компании могут быть просто просуммированы с целью получения показателей для каждого города, а показатели для городов могут быть «свёрнуты» до показателей по отдельным регионам.
- **Нисходящий анализ.** Операция, обратная консолидации, которая включает отображение подробных сведений для рассматриваемых консолидированных данных.

- **Разбиение с поворотом (манипулирование размерностями).** Эта операция позволяет получить представление данных с разных точек зрения. Например, один срез данных о доходах может отображать все сведения о доходах от продаж объектов недвижимости указанного типа по каждому городу. Другой срез может представлять все данные о доходах отделений компании в каждом из городов. Разбиение с поворотом часто выполняется вдоль оси времени с целью анализа тенденций и поиска закономерностей.

Серверы OLAP многомерных баз данных обладают способностью хранить многомерные данные в сжатом виде. Это достигается за счет динамического выбора способа физического хранения данных и технологий сжатия, которые позволяют максимально эффективно использовать имеющееся пространство. **Плотно упакованные данные** (в которых пустые ячейки занимают меньшую часть куба) могут храниться отдельно от **разреженных данных** (в которых пустые ячейки занимают большую часть куба).

Оптимизируя использование пространства, серверы OLAP до минимума сокращают требования, предъявляемые к устройствам физического хранения данных, что, в свою очередь, позволяет эффективно анализировать исключительно большой объем данных. Это также дает возможность загружать больше данных непосредственно в оперативную память компьютера, что приводит к существенному повышению производительности за счет сокращения количества выполняемых операций ввода-вывода.

В конечном итоге предварительное обобщение, использование иерархической структуры размерностей и управление заполнением пространства кубов данных позволяют значительно сократить размер базы данных и исключить потребность многократного вычисления одних и тех же значений. Подобная структура позволяет избежать необходимости соединения нескольких таблиц, а также обеспечивает быстрый и прямой доступ к массивам данных, что существенно ускоряет обработку многомерных запросов.

12 правил Кодда:

1. Многомерное концептуальное представление данных.
2. Прозрачность.
3. Доступность.
4. Неизменно высокая производительность подготовки отчетов.
5. Архитектура «клиент/сервер».
6. Универсальность измерений.
7. Динамическое управление разреженностью матриц.
8. Многопользовательская поддержка.
9. Неограниченные перекрёстные операции между размерностями.
10. Поддержка удобных средств манипулирования данными.
11. Гибкость средств формирования отчётов.
12. Неограниченное число измерений и уровней агрегирования.

24. Категории инструментов OLAP. Системы MOLAP, их устройство, преимущества и недостатки.

Категории инструментов OLAP

Инструменты OLAP классифицируются по архитектуре используемой ими базы данных, содержащей данные для оперативной аналитической обработки. Существуют три основные категории инструментов OLAP:

- MOLAP – Многомерные инструменты OLAP;
- ROLAP – Реляционные (мультиреляционные) инструменты OLAP;
- HOLAP – Гибридные инструменты OLAP, они же – Управляемая среда запросов.

Многомерные OLAP-инструменты – MOLAP

В многомерных OLAP-системах структура куба хранится в многомерной базе данных. В той же базе данных хранятся предварительно обработанные агрегаты и копии листовых (детальных, исходных) значений. В связи с этим все запросы к данным удовлетворяются многомерной системой баз данных, что делает MOLAP-системы исключительно быстрыми.

Для загрузки MOLAP-системы требуется дополнительное время на копирование в многомерную базу всех листовых данных. Поэтому возникают ситуации, когда листовые

данные MOLAP-системы оказываются рассинхронизированными с данными в витрине данных. Иными словами, MOLAP-системы вносят запаздывание в данные нижнего уровня иерархии.

Архитектура MOLAP требует большего объёма дискового пространства из-за хранения в многомерной базе копий листовых данных. Но объём дополнительного пространства обычно не очень велик.

Преимущества MOLAP-систем:

1. Высокая производительность. Все данные хранятся в многомерных структурах, что существенно повышает скорость обработки запросов.
2. Доступны расширенные библиотеки для сложных функций оперативного анализа.
3. Обработка разреженных данных выполняется лучше, чем в ROLAP.
4. Структура и интерфейсы наилучшим образом соответствуют структуре аналитических запросов.
5. Многомерные СУБД легко справляются с задачами включения в информационную модель разнообразных встроенных функций.

Недостатки MOLAP-систем:

1. MOLAP могут работать только со своими собственными многомерными БД и основываются на патентованных технологиях для многомерных СУБД, поэтому являются наиболее дорогими. Эти системы обеспечивают полный цикл OLAP-обработки и либо включают в себя, помимо серверного компонента, собственный интегрированный клиентский интерфейс, либо используют для связи с пользователем внешние программы работы с электронными таблицами.
2. Сложно изменять измерения без повторной агрегации.
3. По сравнению с реляционными, очень неэффективно используют внешнюю память, обладают худшими по сравнению с реляционными БД механизмами транзакций.
4. Отсутствуют единые стандарты на интерфейс, языки описания и манипулирования данными.
5. Не поддерживают репликацию данных, часто используемую в качестве механизма загрузки.

25. Категории инструментов OLAP. Системы ROLAP, их устройство, преимущества и недостатки.

Реляционные OLAP-инструменты – ROLAP

В реляционных OLAP-системах данные хранятся в обычном реляционном формате, но обеспечивается их представление в многомерной форме (кубах данных) через промежуточный слой метаданных. Это позволяет обойтись без создания *статичной* многомерной структуры данных. Благодаря этому упрощается формирование нескольких многомерных представлений одной двумерной таблицы. В одних инструментах ROLAP для повышения производительности обработки данных применяются усовершенствованные машины SQL, обеспечивающие поддержку сложных функций многомерного анализа. В других рекомендуется или требуется наличие денормализованных проектов базы данных, например выполненной по схеме «звезда».

Благодаря реляционной структуре, архитектура ROLAP позволяет хранить большие объемы данных. Поскольку в архитектуре ROLAP листовые значения берутся непосредственно из витрины данных, то возвращаемые ROLAP-системой листовые значения всегда будут соответствовать актуальному на данный момент состоянию предметной области. Иными словами, ROLAP-системы лишены запаздывания в части листовых данных.

Преимущества ROLAP-систем:

1. Возможность использования ROLAP с хранилищами данных и различными OLTP-системами.
2. Реляционные СУБД имеют реальный опыт работы с очень большими БД и развитые средства администрирования. При использовании ROLAP размер хранилища не является таким критичным параметром, как в случае MOLAP.
3. Реляционные СУБД обеспечивают значительно более высокий уровень защиты данных и хорошие возможности разграничения прав доступа.
4. При оперативной аналитической обработке содержимого хранилища данных инструменты ROLAP позволяют производить анализ непосредственно над хранилищем (поскольку в подавляющем большинстве случаев корпоративные хранилища данных реализуются средствами реляционных СУБД).
5. В случае переменной размерности задачи, когда изменения в структуру измерений приходится вносить достаточно часто, ROLAP-системы с динамическим представлением размерности являются оптимальным решением, так как в них такие модификации не требуют физической реорганизации БД, как в случае MOLAP.
6. Системы ROLAP могут функционировать на гораздо менее мощных клиентских станциях, чем системы MOLAP, поскольку основная вычислительная нагрузка в них ложится на сервер, где выполняются сложные аналитические SQL-запросы, формируемые системой.

Недостатки ROLAP-систем:

1. Меньшая производительность, чем у MOLAP. Для обеспечения сравнимой с MOLAP производительности реляционные системы требуют тщательной проработки схемы БД и специальной настройки индексов. Но в результате этих операций производительность хорошо настроенных реляционных систем при использовании схемы «звезда» сравнима с производительностью систем на основе многомерных БД.
2. Функциональность систем ограничивается возможностями SQL, так как аналитические запросы пользователя транслируются в SQL-операторы выборки.
3. Сложно пересчитывать агрегированные значения при изменениях начальных данных.
4. Сложно поддерживать таблицы агрегатов.

26. Категории инструментов OLAP. Системы HOLAP, их устройство, преимущества и недостатки.

Гибридные OLAP-инструменты – HOLAP

В гибридных OLAP сочетаются черты ROLAP и MOLAP. Они разработаны с целью совмещения достоинств и минимизации недостатков, присущих MOLAP и ROLAP.

HOLAP-серверы используют гибридную схему хранения данных, при которой наиболее востребованные агрегированные бизнес-показатели хранятся в многомерном пространстве, а ресурсоёмкие детальные данные — в реляционном.

Серверы MOLAP работают лучше, когда данные более плотные, серверы ROLAP – когда данные разрежены. Серверы HOLAP применяют подход ROLAP для разреженных областей многомерного пространства и подход MOLAP для плотных областей.

Серверы HOLAP разделяют запрос на несколько подзапросов, направляют их к соответствующим фрагментам данных, комбинируют результаты, а затем предоставляют результат пользователю. Материализация выборочных представлений в HOLAP, выборочное построение индексов, а также планирование запросов и ресурсов аналогично тому, как это реализовано в серверах MOLAP и ROLAP.

Используя обе модели (MOLAP и ROLAP), HOLAP предоставляет пользователям быстрый доступ как к подробным данным из реляционной модели, так и к обобщённым данным из многомерной модели.

Данные из СУБД (непосредственно или с помощью сервера MOLAP) могут передаваться на настольный компьютер или локальный сервер в виде «малого» куба данных, который затем может храниться, анализироваться и сопровождаться локально.

Преимущества HOLAP-систем:

1. HOLAP-системы предлагают расширенные возможности хранения данных по сравнению с традиционными OLAP-технологиями. Благодаря гибридной архитектуре HOLAP лучше подходит для работы с большими многомерными базами данных.
2. HOLAP-системы способны выполнять как агрегирование, так и исследование на одном и том же наборе данных. Это преимущество ускоряет выполнение аналитических запросов, поскольку данные доступны в рамках одного кадра. HOLAP также может предложить анализ данных в режиме реального времени, позволяя пользователям быстро получить представление о своих данных.

Недостатки HOLAP-систем:

1. Сложность. Поскольку он требует знания и понимания реляционных и многомерных моделей, HOLAP может оказаться сложным в использовании для нетехнических пользователей. Кроме того, HOLAP требует больше аппаратных ресурсов, чем ROLAP или MOLAP, что приводит к увеличению стоимости этих систем.
2. Эта архитектура приводит к значительному увеличению избыточности данных и может вызвать проблемы при работе в сетях со многими пользователями.

27. Основные модели данных. Иерархическая, сетевая, реляционная модели, их характеристики. Постреляционная модель.

1. Иерархическая модель данных.

Типичный представитель – СУБД IMS (Information Management System) компании IBM (первая версия – 1968 г.) Связи между объектами моделируются иерархией объектов. Модель удобна для описания предметных областей, обладающих изначально

иерархической структурой (например, компания-филиал-отдел-работник). Иерархическая БД состоит из упорядоченного набора деревьев, точнее – из упорядоченного набора нескольких экземпляров одного типа дерева. Тип дерева состоит из одного «корневого» типа записи и упорядоченного набора из нуля или более типов поддеревьев (каждое из которых является некоторым типом дерева). Тип дерева в целом представляет собой иерархически организованный набор типов записи.

2. Сетевая модель данных.

Типичный представитель – СУБД IDMS, разработанная компанией Cullinet Software, Inc. Сетевой подход – это расширение иерархического подхода: в иерархических структурах запись-потомок должна иметь в точности одного предка, в сетевой структуре данных у потомка может иметься любое число предков.

Сетевая БД состоит из набора записей и набора связей между этими записями. Экземпляр типа связи состоит из одного экземпляра типа записи предка и упорядоченного набора экземпляров типа записи потомка. Для данного типа связи *L* с типом записи предка *P* и типом записи потомка *C* должны выполняться следующие два условия:

- каждый экземпляр типа записи *P* является предком только в одном экземпляре типа связи *L*;
- каждый экземпляр типа записи *C* является потомком не более чем в одном экземпляре типа связи *L*.
- **3. Реляционная модель данных.**
- Она на практике оказалась самой жизнеспособной. Все данные – и объекты, и связи – представлены здесь в унифицированной форме: в форме плоских (двумерных) таблиц.
- Эта модель имеет строгое математическое обоснование, основывающееся на теории множеств и логике предикатов – что способствовало созданию декларативного языка SQL, ставшего международным стандартом в отношении операций определения и манипулирования данными.
- В связи с большим объёмом наработок и удобством для многих практических приложений, эта модель активно используется по настоящее время, несмотря на появление более современных моделей данных. Инструментальные средства поддержки реляционной модели данных достаточно просты. **Для полноценного освоения работы в среде реляционной СУБД не обязательно быть специалистом в области программирования.**

4. Постреляционная модель данных

Представляет собой расширенную реляционную модель, в которой отменено требование атомарности атрибутов. Поэтому постреляционную модель называют «не первой нормальной формой» или «многомерной базой данных». Она использует трёхмерные структуры, позволяя хранить в полях таблицы не одно значение, а некоторую **коллекцию значений**. Тем самым расширяются возможности по описанию сложных объектов реального мира. В качестве языка запросов используется несколько расширенный SQL, позволяющий извлекать сложные объекты из одной таблицы без операций соединения.

Самыми известными из коммерческих постреляционных СУБД являются системы: Adabas,Pick,Universe.

28. Объектно-ориентированная и объектно-реляционная модели данных, их основные характеристики.

Объектно-ориентированная модель данных

В этой модели информация представлена в виде объектов, как в объектно-ориентированных языках программирования.

Два альтернативных определения ООСУБД:

- Это есть результат применения идеологии объектно-ориентированного программирования к технологии баз данных;
- Это есть объектно-ориентированная среда программирования, но наделённая возможностью долговременного хранения объектов в компьютерной памяти (среда **перманентного** языка).

Однако в любом случае объектно-ориентированные СУБД объединяют возможности баз данных с возможностями языков объектно-ориентированного программирования.

Примеры объектных СУБД: GemStone, ObjectStore, ITASCA, Caché, Cerebrum, Jasmine.

Парадигма объектной модели данных основывается на ряде базовых понятий, заимствованных из объектно-ориентированного программирования: объект, идентификатор объекта, класс, инкапсуляция, наследование, полиморфизм.

Каждый объект базы данных состоит из:

- **атрибутов** — характеристик, определяющих **состояние** объекта;
- **методов** — запрограммированных действий, оперирующих над состоянием объекта и определяющих его **поведение**.

Объект представляет собой уникально определяемую сущность, которая содержит атрибуты, описывающие состояние какого-либо объекта реального мира, и связанные с этим объектом действия.

Множество объектов с одним и тем же набором свойств (атрибутов и методов) определяет **класс** объекта (некоторый условный аналог схемы реляционной таблицы).

Объектно-реляционная модель возникла как развитие традиционной реляционной модели, за счёт пополнения её рядом новшеств, реализующих основные концепции объектно-ориентированной парадигмы:

1. Расширенный набор типов и возможность на их основе создавать собственные типы, представляющие структуры и коллекции (множества, мультимножества, списки, массивы, словари) и продолжать этот процесс дальше, по индукции.
2. Методы (хранимые процедуры). Позволяется определять специальные процедуры, которые могут выполняться над данными некоторого типа, объявленного пользователем. И хранить эти процедуры наряду с основными данными.
3. Идентификаторы кортежей (аналог OID в объектных системах).
4. Ссылки. Объектно-реляционная модель предоставляет возможность формирования ссылок или указателей на кортежи таблиц.

29. Распределённые базы данных и мультибазовые системы. Их основные характеристики.

Распределённые базы данных

Распределённая база данных (РБД, DDB – Distributed Database) – набор логически связанных между собой разделяемых данных (и их описаний), которые физически распределены в некоторой компьютерной сети.

Распределённая СУБД (РСУБД, DDBMS) – программный комплекс, предназначенный для управления распределёнными БД и позволяющий сделать распределённость информации **прозрачной** для конечного пользователя.

Самый простой вариант распределённой БД: двухзвенная модель клиент-сервер, когда сами данные могут храниться и на сервере, и на клиенте.

При проектировании распределённой базы данных следует ориентироваться на достижение следующих стратегических целей:

1. Локальность ссылок.
2. Повышение надёжности и доступности данных.
3. Приемлемый уровень производительности.
4. Баланс между ёмкостью и стоимостью внешней памяти для хранения данных.
5. Минимизация расходов на пересылку данных.

Гомогенные системы

все сайты находятся под управлением одного и того же типа СУБД

Гетерогенные системы

На сайтах системы могут функционировать различные типы СУБД, использующие (возможно) разные модели данных (иерархическую, сетевую, реляционную, объектную).

Мультибазовые системы

Мультибазовые (интегрированные) системы: СУБД, которые прозрачным образом располагаются поверх уже существующих БД и файловых систем, интегрируя их в единое целое и предоставляя своим пользователям как некую единую базу данных.

В таких СУБД управление каждым из сайтов (**источников** информации) осуществляется совершенно автономно и независимо.

Источники информации для мультибазовой системы в общем случае отличаются гетерогенным характером. В связи с этим в процессе интеграции между данными источников возникает, как правило, серия противоречий, которые надо разрешить при построении мультибазовой системы:

1. Различия в структуре данных.
2. Различия в типах данных.
3. Различие в доменах атрибутов.
4. Семантические различия.

5. Отсутствующие значения.

30. Документоориентированные базы данных, их область применения и преимущества в этих областях.

Документоориентированные базы данных. Подход NoSQL

Документоориентированные базы данных (базы данных документов) – это тип баз данных, направленный на хранение и запрос данных в виде документов, подобном JSON.

JavaScript Object Notation (JSON) – это открытый формат обмена данными, который читается как человеком, так и компьютером.

В отличие от других баз данных, документоориентированные оперируют «документами», сгруппированными по коллекциям. Этот подход является развитием идеи иерархической модели данных.

Документ представляет собой набор атрибутов (ключ и соответствующее ему значение). Значения могут быть как и простыми типами данных (строки, числа или даты), так и более сложными, такими как вложенные объекты, массивы и ссылки на другие документы.

В реляционных базах данных структура записей строго определена и каждая запись содержит одни и те же поля. Даже если поле не используется, оно присутствует, хоть и пустое.

В документоориентированных БД используется другой подход: в них **отсутствует схема данных**, что позволяет добавлять новую информацию в некоторые записи, не требуя при этом, чтобы все остальные записи в базе данных имели одинаковую структуру.

Документы в базе данных адресуются с помощью уникального ключа, обычно это строка, которая генерируется автоматически. По нему можно, например, извлекать запись или ссылаться на другие документы.

Другой значимой особенностью документоориентированных баз данных является то, что помимо простого поиска документов по ключу, как в key-value базах данных, они предоставляют **свой язык запросов**, функционал, синтаксис и производительность которого отличается от одной реализации к другой (подход NoSQL).

Преимущества баз данных документов

Простота разработки. Документы JSON соответствуют объектам – распространенному типу данных в большинстве языков программирования. При разработке приложений разработчики могут гибко создавать и обновлять документы непосредственно из кода. Это означает, что они тратят меньше времени на предварительное создание моделей данных. Таким образом, разработка приложений происходит быстрее и эффективнее.

Гибкая схема. База данных, ориентированная на документы, позволяет создавать несколько документов с разными полями в одной коллекции. Это может быть удобно при хранении неструктурированных данных, таких как электронные письма или публикации в социальных сетях.

Производительность при любом масштабе. Базы данных документов предлагают встроенные возможности распространения. Их можно горизонтально масштабировать на несколько серверов без снижения производительности, что также экономически выгодно. Кроме того, базы данных документов обеспечивают отказоустойчивость и доступность благодаря встроенной репликации.

31. Полуструктурированные базы данных, их основные характеристики.

Полуструктурированные (слабоструктурированные) базы данных

Данные, встречающиеся в реальной жизни, могут иметь значительные различия в степени структурированности.

В традиционных реляционных и объектно-ориентированных базах данных хранятся данные, имеющие строгую и правильную структуру. С другой стороны, фото, аудио и видео-файлы можно отнести к полностью неструктурированным данным. Между этими двумя крайностями находится наибольший объём данных. Такие данные называются полуструктурированными. Примером полуструктурированных данных может быть форматированный текст, HTML-страницы, данные в формате XML и пр.

К **полуструктурированным** относят такие данные, в которых можно выделить некоторую структуру (в этом их отличие от аудио или видео данных), но структура этих данных недостаточно строгая для хранения их в традиционных системах (реляционных, объектно-ориентированных).

Основные особенности полуструктурированных данных:

1. Неявная структура. Многие данные хотя и имеют некоторую достаточно строгую структуру, но эта структура неявная. Например, текстовые документы или HTML страницы на Web-серверах (в этом случае некоторое, но не полное представление о структуре помогут дать теги). Поэтому необходимо разрабатывать методы выявления неявной структуры, таких данных.

2. Частичная структурированность. В некоторых случаях большая часть данных, работу с которыми необходимо автоматизировать, имеют правильную структуру. Тогда для хранения этой части данных используют традиционные системы управления базами данных, и придумывают методы связи этой части данных с оставшимися данными, которые не удалось структурировать и поэтому приходится хранить в других системах (специализированные системы, файловая система и т.д.). Таким образом, системы, построенные по такому принципу, представляют собой надстройку над традиционными СУБД.

3. Частое изменение структуры. Структура полуструктурированных данных часто меняется, что необходимо учитывать при разработке систем, работающих с такими данными.

4. **Апостериорная схема данных.** Традиционные СУБД опираются на принцип фиксированной схемы данных. Поэтому сначала описывают схему базы данных, и только затем можно наполнять ее данными. Системы, использующие подобный подход, можно назвать системами с **априорной схемой**. При работе с полуструктурированными данными целесообразно применять обратный подход: сначала заполняем базу данных, а затем определяем какую структуру (схему) она имеет, то есть при заполнении базы

вырисовывается ее схема. Системы, работающие по последнему принципу, можно назвать системами с **апостериорной схемой**. Использование такого подхода, дает большую гибкость при формировании базы и предоставляет возможность свободно изменять ее структуру. Благодаря применению описанного подхода системами управления полуструктурированными данными, используемую ими модель данных часто называют самоописательной или самооопределяемой.

32. Графовые базы данных, их сравнение с реляционными. Области применения графовых баз данных.

Графовые базы данных (вариант развития сетевой модели).

Графовая база данных – это нереляционная база данных (NoSQL), которая определяют корреляции между сложно взаимосвязанными сущностями. Такая структура позволяет обойти ограничения реляционных БД и уделяет больше внимания отношениям между данными. Графы представляют наборы данных в виде узлов, рёбер и свойств.

- **Узлы** – это экземпляры или сущности данных; ими является любой объект, который планируется к отслеживанию (люди, заказчики, подразделения и пр.);
- **Рёбра** – это важнейшие концепции в графовых БД. Они отображают взаимосвязь между узлами. Имеют направления и могут быть одно- или двунаправленными;
- **Свойства** – содержат описательную информацию, связанную с узлами. В некоторых случаях свойства бывают и у ребер.

Графовые БД предлагают концептуальное представление данных, тесно связанных с реальным миром. Моделировать сложные связи с их помощью гораздо проще, поскольку отношениям между точками данных уделяется такое же внимание, как и самим данным.

Реляционные БД обеспечивают структурированный подход к данным, а графовые считаются более гибкими и ориентированными на быстрое понимание взаимосвязей между данными.

Графовые и реляционные БД имеют свою область применения. Сложные взаимосвязи лучше реализовать через графовые БД, поскольку их возможности превосходят традиционные реляционные СУБД. При создании моделей баз данных в реляционных системах *MySQL* или *PostgreSQL* требуется тщательное планирование, а в графовых используется более естественный и гибкий подход к данным.

Примеры использования графовых баз данных

1. Рекомендательные сервисы в режиме реального времени (например, по продуктам и электронным товарам).
2. Управление цифровыми ресурсами. Объем цифрового контента огромен и постоянно растёт. Графовые БД предлагают масштабируемую и простую модель данных, позволяющую отслеживать цифровые ресурсы: документы, расчеты, контракты и пр.
3. Выявление мошенничества. Поиск подозрительных закономерностей и раскрытие мошеннических платежных схем выполняется в режиме реального времени.
4. Семантический поиск при обработке естественного языка. Семантический поиск помогает определить значение ключевых слов и выдает более подходящие варианты, которые проще отобразить с помощью графовых БД.

5. Сетевое управление. Сети – это связанные графы. Графовые БД снижают время, необходимое для оповещения сетевого администратора о проблемах в сети.