# **33 Подходы к физическому хранению отношений. Построчное хранение отношений. Понятие tid-а.**

Возникают следующие разновидности объектов во внешней памяти базы данных:

1. Отношения – основная часть базы данных
2. Служебные отношения (хранят информацию о схеме, параметрах ключа и т.д.)
3. индексы - управляющие структуры, ускоряющие доступ
4. журналы (надежность хранения данных)
5. служебная информация, поддерживаемая для внутренних потребностей нижнего уровня системы (например, информация о свободной памяти).

Хранение Отношений:

1. Покортежное или построчное (быстрый доступ, занимает больше места).

Внешняя память – множество сегментов, разбитых на страницы.

tid – идентификатор кортежа (обычно, <№страницы, № кортежа>).

В случае перемещения кортежа меняется описание, но tid сохраняется.

1. Постолбцовое (сохранение места, но трудоемкая сборка). Редко применимо и дальше не рассматривается.

34, 35 Понятие индексов в базе данных. Техника хранения на основе b-деревьев. Методы хеширования.

Основное назначение индексов состоит в обеспечении эффективного прямого доступа к кортежу таблицы по ключу.

Обычно индекс определяется для одной таблицы, и ключом является значение ее поля. Если ключом индекса является возможный ключ таблицы, то индекс должен обладать свойством уникальности, т.е. не содержать дубликатов ключа.

Полезным свойством индекса является обеспечение последовательного просмотра кортежей таблицы в заданном диапазоне значений ключа в порядке возрастания или убывания значений ключа.

Механизм мульти-индекса… Никому не нужен! Если нужен, то вот… используется для оптимизации эквисоединения таблиц. Суть в том что мульти-индексы организуются для таблиц с одинаковыми атрибутами (один или несколько из атрибутов становится ключом мульти-индекса). Значению ключа сопоставляется набор кортежей этих таблиц, значения выделенных атрибутов которых совпадают со значением ключа.

Общей идеей любой организации индекса (для прямого доступа по ключу и просмотра в порядке возрастания/убывания значений ключа) является хранение упорядоченного списка значений ключа (со списком идентификаторов кортежей для каждого значения ключа). Одна организация индекса отличается от другой, главным образом, в способе поиска ключа с заданным значением.

**B-деревья** – способ организации индексов, дерево во внешней памяти (накладные расходы велики, менять информацию легче)

Механизм B-деревьев (B+) ориентирован на хранение во внешней памяти 2 типов страниц:

1. Внутренних – это последовательность значений ключей и ссылка на № следующей страницы, либо на листовую.



При этом выдерживаются следующие свойства:

i. ключ1 < ключ2 <...< ключm

ii. в странице дерева Nm находятся ключи k со значениями ключm <= k <= ключm+1.

1. Листовых



i. ключ1 < ключ2 < ... < ключk;

ii. списокr – упорядоченный список идентификаторов кортежей (tid), включающих значение ключr;

iii. листовые страницы связаны одно- или двунаправленным списком.

B-дерево – сбалансировано (ветви до листьев одинаковой глубины).

Глубина ~logmN, где m – число ключей во внутренней странице, N – число кортежей.

При создании дерева, в случае переполнения порождается новая страница, в которую переливается половина предыдущей. => частые расщепления и слияния.

Решение:

1. Упреждения расщепления (раннее расщепление по достижении некого уровня)
2. Переливание (равновесие соседних вершин)
3. Слияние 3-в-2 (а не 2-в-1)

**Хеширование**– способ организации индексов.

1. Классический случай – свертка ключа используется как адрес в таблице, содержащей ключи и записи. Основным требованием к хэш-функции является равномерное распределение значение свертки
2. Расширяемое хэширование (Extendible Hashing) – принцип использования деревьев цифрового поиска в основной памяти. В основной памяти поддерживается справочник, организованный на основе бинарного дерева цифрового поиска, ключами которого являются значения хэш-функции, а в листовых вершинах хранятся номера блоков записей во внешней памяти.
3. Линейное хэширование (Linear Hashing) – является то, что для адресации блока внешней памяти всегда используются младшие биты значения хэш-функции. Если возникает потребность в расщеплении, то записи перераспределяются по блокам так, чтобы адресация осталась правильной.

**36. Интерфейс ядра SQL-ориентированных СУБД - RSS.**

В RSS нет именования объектов базы данных как в SQL, используются уникальные идентификаторы. Операции включают сканирование, создание/уничтожение объектов, модификацию, добавление поля к отношению, управление транзакциями, синхронизацию. Сканирование может быть по отношению, по индексу или списку. При сканировании по индексу, порядок соответствует порядку значений ключа индекса. При сканировании списка, порядок совпадает с порядком занесения кортежей в список. После открытия сканирования, возвращается идентификатор сканирования.

Операция NEXT читает следующий кортеж сканирования, удовлетворяющий условиям. Условие - это дизъюнктивная нормальная форма простых условий, относящихся к значениям полей отношения. После достижения правой границы диапазона сканирования, следует закрытие сканирования - операция CLOSE.

Операция CLOSE может быть выполнена в любом открытом сканировании. Параметром операции является идентификатор сканирования, который становится недействительным после выполнения операции.

Операции создания и уничтожения объектов базы данных включают создание таблиц, списков, индексов и уничтожение любого из этих объектов. Параметром операций создания является спецификатор структуры объекта. В результате выполнения этих операций создается описатель и вырабатывается идентификатор объекта.

Операция CREATE IMAGE создает индекс для таблицы. Параметрами являются идентификатор таблицы, список номеров полей, составляющих ключ индекса, и признаки упорядочения. Может быть указан признак уникальности индекса. Если операция выполняется по отношению к пустой таблице, то создается описатель в служебном отношении описателей индексов и возвращается идентификатор индекса.

Если таблица не пуста при создании индекса, операция становится дорогой, так как требуется создание B-дерева индекса. Если индекс уникален и уникальность нарушается, операция не выполняется.

Операции DROP TABLE, DROP LIST и DROP IMAGE могут быть выполнены в любое время, приводя к уничтожению объекта и недействительности его идентификатора.

Массовые операции над постоянными объектами требуют дополнительных расходов из-за необходимости обеспечения возможности откатов транзакции. Уничтожение непустых таблиц - дорогая операция.

Операции модификации включают вставку кортежа в отношение или список, удаление кортежа из отношения и модификацию кортежа. Параметры операции вставки - идентификатор отношения или списка и набор значений полей кортежа. При вставке кортежа в список, он помещается в конец списка.

При вставке кортежа в постоянное отношение корректируются все индексы. Если индекс уникален и уникальность нарушается, операция вставки считается невыполненной.

После успешной вставки кортежа в отношение, вырабатывается tid нового кортежа. При вставке кортежа в список, tid не вырабатывается.

Операции удаления и модификации кортежей допускаются только для кортежей постоянных таблиц. Идентификация кортежа может быть с помощью tid’а кортежа или идентификатора сканирования.

Операция DELETE требует идентификатор кортежа. Операция UPDATE требует идентификатор кортежа и спецификацию изменяемых полей кортежа. Обе операции могут повлечь коррекцию индексов.

**37. Этапы проектирования баз данных. Ограниченность реляционной модели при использовании в проектировании. Понятие концептуальной (семантической, инфологической) модели. Достоинства концептуальных моделей. Средства автоматизации проектирования баз данных.**

Проектирование БД.

|  |
| --- |
| Концептуальное |

П роцесс создания модели, использование информации, не зависящей от любых физических аспектов ее представления.

|  |
| --- |
| Логическое |

П роцесс создания модели, используемой по предполагаемой информации с учетов выбранной модели организации данных, но независящей от типа целевой СУБД и др. физических аспектов ее реализации.

|  |
| --- |
| Физическое |

Процесс описания реализации БД с учетом выбранной целевой СУБД и указанием структур хранения и методов доступа, используемых для организации эффективной обработки данных.

Ограниченность проектирования:

1. Реляционная модель данных не предлагает какого-либо механизма для разделения сущностей объектов предметной области и связей между ними,
2. Модель не обеспечивает достаточных средств для представления смысла данных,
3. Во многих прикладных областях трудно моделировать предметную область на основе плоских таблиц,
4. Хотя весь процесс проектирования происходит на основе учета зависимостей, реляционная модель не предоставляет какие-либо формализованные средства для представления этих зависимостей,
5. На ранних стадиях требуется участие специалистов данной ПО.

Информационная модель – способ представления понятий или объектов ПО, описание сущностей для данного предприятия совокупности объектов, их параметры, поведение и отношение между ними.

Достоинства информационного моделирования:

1. Построение мощной и наглядной концептуальной схемы БД позволяет более полно оценить специфику моделируемой предметной области и избежать возможных ошибок на стадии проектирования схемы реляционной БД
2. Информационная модель в виде документации (хотя бы в виде вручную нарисованных диаграмм и комментариев к ним), которая может оказаться очень полезной не только при проектировании схемы реляционной БД, но и при эксплуатации, сопровождении и развитии уже заполненной БД.
3. На рынке представлены CASE-системы, обеспечивает автоматизированное преобразование диаграммных концептуальных схем баз данных, представленных в той или иной семантической модели данных, в реляционные схемы, специфицированные чаще всего на языке SQL

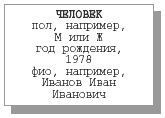
CASE – computer aided software engineering.

Система поддержки технической автоматизации проектрирования, реализации и сопроводления сложных тпрограммных систем на всех этапах их жизненного цикла

**38. ER-модель. Основные понятия. Представление на диаграммах сущностей, атрибутов и связей. Примеры. Уникальные идентификаторы типов сущностей. Нормальные формы ER-моделей.**

Entity – Relation модель (сущность - связь) предназначается для описания ПО в целях проектирования БД(моделирование базовых понятий на простых графиках и диаграммах).

**Основные понятия:**

**С****ущность** - это реальный или представляемый объект, информация о котором должна сохраняться и быть доступной (изображается прямоугольником)

**Атрибут (сущности)**– любая деталь, которая служит для уточнения, идентификации, классификации, численной характеристики или выражения состояния сущности.

(Пример справа)

**Связь** – это графически изображаемая ассоциация, устанавливаемая между двумя сущностями.

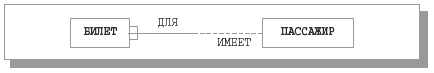
ER-модели связи бинарные. В любой связи выделяются два конца, на каждом из которых указываются *имя* конца связи, *степень* конца связи (сколько экземпляров данного типа сущности должно присутствовать в каждом экземпляре связи), *обязательность* связи.

Связь представляется в виде ненаправленной линии, соединяющей две сущности или ведущей от сущности к ней же самой. При этом в месте «стыковки» связи с сущностью используются *трехточечный* вход (в связи могут/должны использоваться много экземпляров сущности) либо *одноточечный* вход(может/должен участвовать только один экземпляр сущности).

Обязательный конец связи изображается сплошной линией, а необязательный – прерывистой линией.

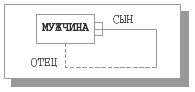
Примеры.

«Многие к одному»



* каждый БИЛЕТ предназначен для одного и только одного ПАССАЖИРА;
* каждый ПАССАЖИР может иметь один или более БИЛЕТОВ.

«Рекурсивная модель»



* каждый МУЖЧИНА является сыном одного и только одного МУЖЧИНЫ;
* каждый МУЖЧИНА может являться отцом одного или более МУЖЧИН.

Как отмечалось выше, при определении типа сущности необходимо гарантировать, что каждый экземпляр сущности является отличимым от любого другого экземпляра той же сущности. Необходимо сообщить системе автоматизации проектирования БД, каким образом будут различаться эти экземпляры, т. е. сообщить, как конструируются уникальные идентификаторы экземпляров каждого типа сущности.

**Уникальным идентификатором сущности** может быть атрибут, комбинация атрибутов, связь, комбинация связей или комбинация связей и атрибутов, уникально отличающая любой экземпляр сущности от других экземпляров сущности того же типа.

# **Получение реляционной схемы из er-диаграммы. Пошаговый алгоритм (без учета наследования и взаимно исключающих связей).**

1. Каждый **простой тип сущности** превращается в таблицу.

Имя сущности – имя таблицы. Экземпляр – строка.

Простым типом сущности называется тип сущности, не являющийся подтипом и не имеющий подтипов (см. опр. в Б36).

1. Каждый **атрибут** => столбец таблицы
2. Компоненты **уникального идентификатора сущности** превращаются в первичный ключ таблицы.

Если в состав уникального идентификатора входят связи, к числу столбцов первичного ключа добавляется копия уникального идентификатора сущности, находящейся на дальнем конце связи

1. **Связи «многие к одному»** (и «один к одному») становятся внешними ключами, т.е. образуется копия уникального идентификатора сущности на конце связи «один», и соответствующие столбцы составляют внешний ключ таблицы, соответствующей типу сущности на конце связи «многие».

Необязательные связи соответствуют столбцам внешнего ключа, допускающим наличие неопределенных значений; обязательные связи – не допускают.

Если между двумя типами сущности A и B имеется связь «один к одному», то соответствующий внешний ключ по желанию проектировщика может быть объявлен как в таблице A, так и в таблице B. Чтобы подчеркнуть единственность в определении таблицы, соответствующий столбец должен быть специфицирован как возможный ключ таблицы.

1. **связь «многие ко многим»** между типами сущности A и B создается дополнительная таблица AB с двумя столбцами, один из которых содержит уникальные идентификаторы экземпляров сущности A, а другой – уникальные идентификаторы экземпляров сущности B.

## Наследование сущностей в er-модели. Примеры. Отображение диаграммы с наследованием в реляционную схему.

Тип сущности может быть расщеплён на несколько **подтипов**, каждый из которых включает общие атрибуты и/или связи. Тип сущности, на основе которого определяются подтипы, называется **супертипом**.

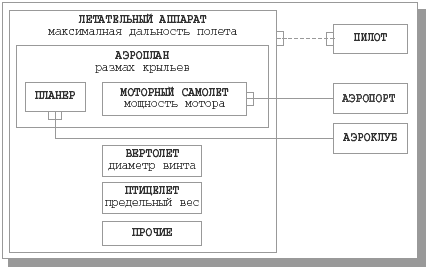
ER-модели не ограничивает подтипизацию, обычно оказывается достаточно 2-3 уровней.

**Простой тип** – не является подтипом и не имеет подтипов.

Особенности механизма наследования:

1. Включение: для любого b Є Bi => b Є А
2. Отсутствие собственных экземпляров у супертипа: для любого a Є A => a Є Bi.
3. Разъединенность подтипов: для любого b Є Bi => b !Є Bj.

Пример:



**Супертипы и подтипы.**

## Общая таблица для всех подтипов

A

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| T | a1 | a2 | … | … | … | aM | b11 | … | … | … | b1k1 | … | … | … | bnkn |
| ‘B1’ | X | X | X | X | X | X | X | X | X | X | X | NULL | NULL | NULL | NULL |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

T-признак подтипа, X-некоторое значние.

Для супертипа верно что (см след билет):

1. для любого b Є Bi => b Є А
2. для любого a Є A => a Є Bi
3. для любого b Є Bi => b !Є Bj

Извлечь объекты супертипа: PROJECT A{a1…am, bi1…bik}WHERE t =’Bi’

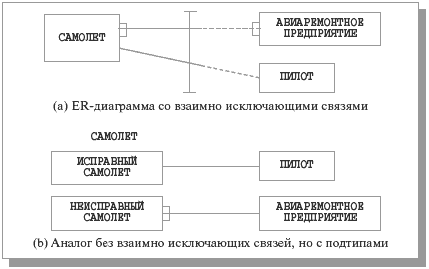
## Взаимно исключающие связи в er-модели. Примеры. Отображение диаграммы со взаимно исключающими связями в реляционную схему.

Для заданной сущности можно определить такой набор связей с другой сущностью, что для каждого экземпляра сущности может/должен существовать только одной связи из данного набора.

Диаграмма со взаимно исключающими связями может быть преобразована к диаграмме с наследованием 2 способами:

1. Специализация (внедрение подтипов). Пример, см рис Б.
2. Введение общего супертипа (обобщение).

Так чтобы у заданной сущности была только одна связь с этим супертипом (который содержал бы в себе остальные экземпляры сущностей).



1. Представление взаимоисключающих связей.

Преобразовать взаимоисключающие связи по пункту 6.

Если связь 1-мн (причем конец много подсоединен к сущности), то следующие выходы:

1. **общее хранение внешних ключей**;

Если внешние ключи всех потенциально связанных таблиц имеют общий формат, то можно применить способ (a), т. е. создать два столбца: идентификатор связи и идентификатор сущности (возможно, составной).

1. **раздельное хранение внешних ключей**.

В таблице как минимум n столбцов (любой может быть составным – внешний ключ связи)

Достоинства а.

* минимальное число столбцов

Недостатки а.

* усложнение выполнения операции соединения

Достоинства б.

* Упрощение операции соединение (A *NATURAL JOIN* B)

Недостатки б.

* Требуется большое число столбцов (в любом может быть NULL) => расход памяти.
  1. . Диаграммы классов языка uml. Основные понятия. Отображение классов, стереотипов, комментариев и ограничений на диаграммах. Примеры.

UML позволяет моделировать разные виды систем: чисто программные, чисто аппаратные, программно-аппаратные, смешанные, явно включающие деятельность людей и т. д.

Стандарт: 12 диаграмм классов, позволяющих описать статические (структурные) и динамические (поведенческие) свойства систем.

Основные понятия:

Диаграмма классов может включать **комментарии** (соединяются пунктиром с той сущностью, которую поясняют) и **ограничения** (заключают в { }). Ограничения могут неформально задаваться на естественном языке или же могут формулироваться на языке объектных ограничений OCL (Object Constraints Language).

**Классом** называется именованное описание совокупности объектов с общими атрибутами, операциями, связями и семантикой (изображается прямоугольником).

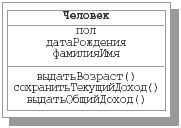
**Атрибутом**класса называется именованное свойство класса, описывающее множество значений, которые могут принимать экземпляры этого свойства. Атрибут является абстракцией состояния объекта.

**Операцией** класса называется именованная услуга, которую можно запросить у любого объекта этого класса. Операция – это абстракция поведения объекта.

**Стереотип** – механизм расширения семантики UML, позволяющий создать новые элементы UML на основе существующих, с учетом особенности решения задачи.

В стандарте предусмотрен набор стереотипов.

*Класс*

*Стереотип*

|  |
| --- |
| <<enumeration>>  Sex |
| М |
| Ж |

|  |
| --- |
| //Определение  перечисляемого типа |

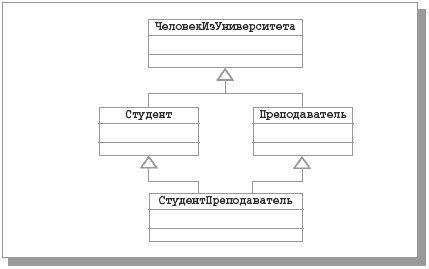
# **Диаграммы классов языка uml. Категории связей и их отображение на диаграмме. Примеры.**

**Связью-обобщением** называется связь между общей сущностью (суперкласс/родителем) и более специализированным типом этой сущности (подкласс/потомок).

*Полиморфизмом по включению*– объекты потомка могут использоваться везде, где могут использоваться объекты предка. Потомок наследует все атрибуты и операции предка.

У класса есть скрытое свойство *isAbstract* – может ли класс иметь объекты.

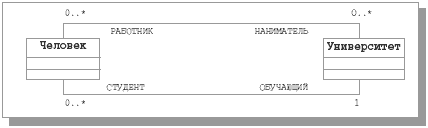
**Множественное наследование** – подкласс определяется на основе нескольких суперклассов.



**Ассоциацией** называется структурная связь между объектами одного класса и объектами другого или того же самого класса (изображается линией).

В ассоциации могут связываться два класса, и тогда она называется бинарной. Допускается создание ассоциаций, связывающих n классов – n-арные ассоциации.

С ассоциацией связаны 5 важных понятий: *имя, роль, кратность, агрегация и навигация*.

У связи может быть **имя**, треугольник указывает направление чтения имени связи; у каждого класса может быть **роль** в ассоциации;

**кратность**(задается числом, интервалом, списком значений и интервалов, \* - бесконечность) показывает, сколько объектов класса может/должно участвовать в экземпляре ассоциации.

**А****грегация** в UML используется для отображения связи «часть-целое» (класс-целое имеет более высокий концептуальный уровень).

Иногда связь «части» и «целого» настолько сильна, что «часть» может быть частью только 1 «целого», и уничтожение «целого» приводит к уничтожению всех его «частей». Такие агрегации– **композитные**.

**Н****авигация** может осуществляться в обоих направлениях. Для некоторых ассоциаций желательно ограничить навигацию (стрелка).

С точки зрения библиотеки разумно произвести навигацию от объекта-библиотеки к связанным с ним объектам-книгам. Однако вряд ли потребуется по данному экземпляру книги узнать, в какой библиотеке она находится

* 1. . Язык ocl. Инварианты ocl. Основные типы данных и выражения.

В UML 2 способа определения ограничений: на естественном языке и на языке OCL. Более точный и лаконичный способ формулировки ограничений обеспечивает язык OCL (Object Constraints Language).

Из языка UML в OCL заимствованы, в первую очередь, следующие концепции:

* класс, атрибут, операция;
* объект (экземпляр класса);
* ассоциация;
* тип данных (включая набор типов Boolean, Integer, Real и String);
* значение (экземпляр типа данных).

В OCL предопределены структурные типы, которые являются разновидностями типов коллекций (collection):

* математическое множество **(set),** неупорядоченная коллекция, нет одинаковых элементов;
* мультимножество **(bag),** неупорядоченная коллекция, которая может содержать элементы-дубликаты;
* последовательность **(sequence),** упорядоченная коллекция, которая может содержать элементы-дубликаты.
* упорядоченное множнество **(ordered set),** упорядоченная коллекция, нет одинаковых элементов;

OCL может применяться для определения ограничений, описывающих пред- и постусловия операций классов, и ограничений, представляющих собой инварианты классов.

**Инвариант класса** – это логическое выражение, вычисление которого должно давать true при создании любого объекта данного класса и сохранять истинное значение в течение всего времени существования этого объекта.

Синтаксис определения инварианта:

context <class\_name> inv:

<OCL-выражение>

Пример (существует человек, обладающий контрольным пакетом акций)

Context Компания inv:

Self.акционер =>Exists

(a:человек|a.количествоакций >= self.количествоакций div 2 +1)

**Операции над объектами**

<объект>.<имя\_атрибута> (значение атрибута)

<объект>.<имя\_роли> (переход по соединению, под именем имеется ввиду имя роли объекта на другом конце связи)

<объект>.<имя\_операции>(фактический параметр, т.е. можно вызвать операцию)

**Операции над коллекциями**

<коллекция><имя операции> (<список фактических параметров>)

|  |  |  |  |
| --- | --- | --- | --- |
| Select(iterat/логическое выражение) | | Результат – новая коллекция (из старых элементов, удовлетворяющих логическому выражению) | |
| collect(iterat/выражение) | | Результат – новая коллекция (элементы вычисляются по выражению) | |
| exists(iterat/логическое выражение) | | True, если выполнено хоты бы для 1 элемента | |
| forAll(iterat/логическое выражение) | | True, если выполнено для всех элементов | |
| Size() | | Число элементов | |
| union, intersect, symmetricDifference | |  | |
| At(index) | |  | |
| Count(element) | |  | |
| Includes/excludes(element) | |  | |
|  |  | | |
|  | | |  |
|  | | |  |
|  | | |  |
|  | | |  |

|  |  |
| --- | --- |
| **Тип** | **Операции** |
| Boolean | And or xor implies if\_then\_else |
| Integer | +-\*/ div mod min max сравнение |
| Real | +-\*/ abs float round min max сравнение |
| String | Concat size substring toUpper toLower |

# **Получение реляционной схемы из диаграммы классов. Основные проблемы и рекомендации.**

Рекомендации:

* не увлекайтесь определением операций в классах
* старайтесь избегать множественное наследие и осторожно использовать одиночное
* реализ. в СУБД с точно заданными кратностями возможно, но предпочтит. доп. триггеров и ограничений (уменьшает эффективность)
* агрег. ассоциации не естественны, компоретные (ПОПРАВЬТЕ, НЕ РАЗОБРАЛ) влияют только на способ поддержки ссылочной целостности
* однонаправленные связи естественны только для объектно-ориентированных СУБД. Для реляционных БД поддержка вызывает дополнительные накладные расходы, значит, снижается эффективность.
* не злоупотребляйте ограничениями (снижается эффективность)
  1. Язык баз данных SQL. Возможности и структура языка SQL. Основные черты модели данных SQL и ее отличия от реляционной модели. Критика SQL.В стандартах SQL определяется собственная модель данных, она похожа на реляционную, но значительно отличается. Итак, 2 важных отличия:
* В SQL данные – это *набор таблиц*, каждая таблица содержит *мультимножество* строк, соответствующих заголовку таблицы. В реляционной модели “фундаментальная абстрактная «родовая» структура данных отношение, представляет собой множество кортежей.” (*прим.Валеры. Я сам не понял смысла, извините*)
* В SQL для таблицы поддерживается порядок столбцов, соответствующий порядку их определения. В реляционной модели атрибуты отношения не упорядочены.

Другими словами, таблица – это вовсе не отношение, хотя во многом они похожи.

Отмечается, что из этого следует, что:

* в SQL отсутствует обязательное предписание об ограничении целостности сущности.
* В базе данных могут существовать таблицы, для которых не определен первичный ключ.

С другой стороны, если для таблицы определен первичный ключ, то для нее ограничение целостности сущности поддерживается так же как и в реляционной модели данных.

Ссылочная целостность в модели данных SQL поддерживается в обязательном порядке, но в трех разных вариантах, лишь один из которых полностью соответствует реляционной модели. Это связано с интенсивным использованием в SQL неопределенных значений.

**Стандарт SQL:2003 – основные тома:**

* 9075-1, SQL/Framework;
* 9075-2, SQL/Foundation;
* 9075-3, SQL/CLI;
* 9075-4, SQL/PSM;
* 9075-9, SQL/MED;
* 9075-10, SQL/OLB;
* 9075-11, SQL/Schemata;
* 9075-13, SQL/JRT;
* 9075-14, SQL/XML.

Структура языка SQL

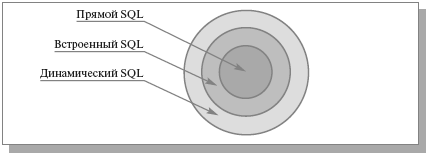
*1.*Классификация, ориентированная на *производителей* СУБД (изнутри). Здесь каждая компания может выбрать уровни, которые они реализуют для своей БД. **Язык разбивается на**базовый (entry)**,**промежуточный(intermediate)**и**полный (full)**уровни.**

*2.*Классификация, ориентированная на программиста приложений БД(снаружи):

Прямой – например,интерактивная консоль(вы пишете SELECT \* FROM… , и вам сразу выдают результат)

встраиваемый – например, реализация библиотек для языков программирования, что бы можно было в этих программах использовать SQL,

динамический – здесь сам SQL запрос может динамически формироваться по ходу программы(в предыдущем сама “строка SQL запросы” была как бы “постоянна”).



# **Основные типы данных языка sql (без учета объектных расширений). Преобразования типов данных**

Все данные, хранящиеся в таблицах, типизированы. Каждому столбцу определяемой таблицы приписывается свой собственный тип. СУБД должна следить за соответствием типов (допустимыми значениями).

**Типы данных**

**Логические типы**(TRUE, FALSE, UNKNOWN). В стандарте UNKNOWN и NULL не различаются.

**Точные числовые типы** (SMALLINT, INTEGER). С дробной частью – NUMERIC (p,s), DECIMAL(p,s). P – число десятичных цифр, s – шкала (число десятичных цифр в дробной части). Для p,s возможны значения по умолчанию. Для NUMERIC р – строгое, для ), DECIMAL – максимально возможное. Пример: 1234,5 – в NUMERIC недопустимо, в ), DECIMAL допустимо.

**Приближенные числовые типы**(вещественные типы REAL, DOUBLE PRECISION, FLOAT(p)). Параметр р – число значимых цифр в мантиссе.

**Символьные строки***.*CHAR(х) – строки фиксированной длины х (по умолчанию х = 1), VARCHAR – произвольной длины, но не более х (по умолчанию х = 1). CLOB (character large object) – этот тип данных предназначен для определения столбцов, хранящих большие и разные по размеру группы символов. При определении столбца задается спецификация CLOB (z), где z задает максимальный размер соответствующей группы символов. Максимально возможное значение параметра z определяется в реализации, но, очевидно, что оно должно быть существенно больше максимально возможного значения параметра x, присутствующего в типах CHAR и CHAR VARYING (z >> x; если длина меньше х, то справа добавляются пробелы).

**Битовые строки**(BIT, BIT VARYING(х), BLOB(x)).

**Дата и время.**DATE : “YYYY-MM-DD”, TIME “HH:MM-SS:FF” (число символов в FF определяется параметром р), TIMESTAMP “YYYY-MM-DD HH:MM-SS:FF”(по умолчанию р = 6, для секунд допустим диапазон 0..61 ), TIME WITH TIME ZONE – +HH:MM, -HH:MM по Гринвичу. Стандартные функции: CURRENT\_DATE(\_TIME, \_STAMP).

**Временные интервалы** ( INTERVAL ‘start(p)[TO end (q)]’). Р—точность (по умолчанию p = 2), разница не привязана к началу/концу интервала.

*( прим. остальных типов в лекциях Маши нет)*

**Преобразование типов.**В SQL поддерживаются явные и неявные преобразования значений одного типа к значениям другого. Неявные (интуитивные) преобразования типов не всегда удобны, недостаточно гибки и иногда могут вызывать ошибки. Поэтому, как показывает предыдущий подраздел, число допустимых неявных преобразований типов в SQL весьма ограничено. Однако в SQL существует специальный оператор CAST, с помощью которого можно *явно* преобразовывать типы или домены в более широких пределах допускаемых преобразований. Конструкция имеет следующий синтаксис:

CAST ({scalar-expression | NULL } AS

{data\_type | domain\_name})

## Средства работы с доменами в sql.

Домен является долговременно хранимым, именованным объектом схемы базы данных. Домены можно создавать (определять), изменять (изменять определения) и ликвидировать (отменять определение). Имена доменов можно использовать при определении столбцов таблиц**. Можно считать, что в SQL  *определение домена* представляет собой вынесенное за пределы определения индивидуальной таблицы «родовое» определение столбца, которое можно использовать для определения различных реальных столбцов реальных базовых таблиц**.

# **Определение домена.**

Для определения домена в SQL используется оператор CREATE DOMAIN. Общий синтаксис этого оператора следующий:88)

domain\_definition ::= CREATE DOMAIN domain\_name [AS] data\_type

[ default\_definition ]

[ domain\_constraint\_definition\_list ]

Здесь domain\_name задает имя создаваемого домена89), data\_type есть спецификация определяющего типа данных. В необязательных разделах default\_definition и domain\_constraint\_definition\_list специфицируются значение домена по умолчанию90) и набор ограничений целостности, которые будут применяться к любому столбцу, определенному на этом домене.

Раздел default\_definition имеет вид

DEFAULT { literal | niladic\_function | NULL }91)

Здесь literal представляет любое допустимое литеральное значение определяющего типа домена, NULL обозначает неопределенное значение, а niladic\_function может задаваться в одной из следующих форм:

USER

CURRENT\_USER

SESSION\_USER

SYSTEM\_USER

CURRENT\_DATE

CURRENT\_TIME

CURRENT\_TIMESTAMP92)

Если в операторе CREATE DOMAIN значение по умолчанию не специфицируется, считается, что такого значения нет. Однако позже к определению домена можно добавить раздел значения по умолчанию с помощью оператора ALTER DOMAIN. Кроме того, этот оператор позволяет удалить раздел значения по умолчанию из существующего определения домена.

Элемент списка domain\_constraint\_definition\_list имеет вид

[CONSTRAINT constraint\_name]

CHECK (conditional\_expression)

Необязательный раздел CONSTRAINT constraint\_name позволяет определить имя нового ограничения целостности. Если явное указание имени отсутствует, ограничению назначается имя, автоматически генерируемое системой. Что касается вида условного выражения, служащего собственно ограничением целостности, то в стандарте запрещается лишь прямое или косвенное использование в нем домена, в определение которого входит данное условное выражение.93) Однако наиболее естественным (и наиболее распространенным) видом ограничения домена является следующий:

CHECK (VALUE IN (list\_of\_valid\_values))

Такое ограничение запрещает появление в любом столбце, определенном на данном домене, любого значения определяющего типа, не входящего в список допустимых значений.

**Изменение домена.**

Для изменения характеристик ранее определенного домена используется оператор SQL  ALTER DOMAIN. Синтаксис этого оператора выглядит следующим образом:

domain\_alternation ::=

ALTER DOMAIN domain\_name domain\_alternation\_action

domain\_alternation\_action ::=

domain\_default\_alternation\_action

| domain\_constraint\_alternation\_action

Как видно из синтаксических правил, при изменении определения домена можно выполнить действие по изменению раздела значения по умолчанию либо изменить ограничение домена. Для первого варианта действует следующий синтаксис:

domain\_default\_alternation\_action ::=

SET default\_definition

| DROP DEFAULT

В случае установки нового значения по умолчанию (SET) это значение автоматически применяется ко всем столбцам, определенным на данном домене. Более точно, это значение становится новым значением по умолчанию. Операция не оказывает влияния на состояние существующих строк таблиц базы данных. В случае отмены раздела значения по умолчанию в определении домена (DROP) существовашее значение домена по умолчанию становится значением по умолчанию каждого столбца, который определен на данном домене и для которого не специфицировано собственное значение по умолчанию.

Действие по изменению ограничения домена определяется следующим синтаксисом:

domain\_constraint\_alternation\_action ::=

ADD domain\_constraint\_definition

| DROP CONSTRAINT constraint\_name

Действие по добавлению нового определения ограничения домена (ADD) приводит к тому, что новое условие добавляется через AND к существующему ограничению домена. Если к моменту выполнения соответствующего оператора ALTER DOMAIN существуют столбцы некоторых таблиц, текущие значения которых противоречат новому ограничению, то СУБД должна отвергнуть этот оператор ALTER DOMAIN. Действие по отмене ограничения домена (DROP) приводит к исчезновению соответствующей части общего ограничения соответствующего домена, что, естественно, не влияет на существующие значения столбцов имеющихся таблиц.

**Отмена домена.**

Чтобы отменить ранее созданное определение домена, нужно воспользоваться оператором DROP DOMAIN в следующем синтаксисе:

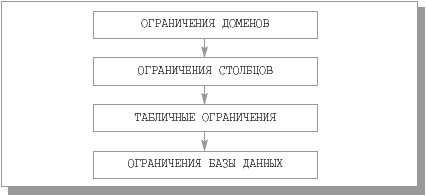
DROP DOMAIN domain\_name {RESTRICT | CASCADES}

# **Средства определения, изменения и отмены определения базовых таблиц в sql.**

Определение базовой таблицы.

Определение столбца: возможно использование определенного ранее домена.

Определение ограничений целостности: для столбца, для таблицы, для всей базы. Ограничения первичного ключа, ограничения внешнего ключа.



Пример:

(1) CREATE TABLE EMP (

(2) EMP\_NO EMP\_NO PRIMARY KEY,

(3) EMP\_NAME VARCHAR(20) DEFAULT 'Incognito' NOT NULL,

(4) EMP\_BDATE DATE DEFAULT NULL CHECK (

VALUE >= DATE '1917-10-24'),

(5) EMP\_SAL SALARY,

(6) DEPT\_NO DEPT\_NO DEFAULT NULL REFERENCES

DEPT ON DELETE SET NULL,

(7) PRO\_NO PRO\_NO DEFAULT NULL,

(8) FOREIGN KEY PRO\_NO REFERENCES PRO (PRO\_NO)

ON DELETE SET NULL,

(9) CONSTRAINT PRO\_EMP\_NO CHECK

((SELECT COUNT (\*) FROM EMP E

WHERE E.PRO\_NO = PRO\_NO) <= 50));

* 1. **Иерархия ограничений в SQL. Средства определения и отмены общих ограничений (ограничений БД).**

**Ограничения SQL** — это правила, применяемые к столбцам данных таблицы. Они используются, чтобы ограничить типы данных, которые могут храниться в таблице.

Ограничения могут применяться либо на уровне столбцов, либо на уровне таблицы.

Вот некоторые из наиболее часто используемых ограничений:

* **NOT NULL Constraint** — столбец не может иметь значение NULL.
* **DEFAULT Constraint** — задаёт значение по умолчанию для столбца, если оно не указано.
* **UNIQUE Constraint** — все значения в столбце должны быть разными.
* **PRIMARY Key** — уникальная идентификация каждой строки или записи в таблице базы данных.
* **FOREIGN Key** — уникально идентифицирует строку или запись в любой другой таблице базы данных.
* **CHECK Constraint** — обеспечивает, чтобы все значения в столбце удовлетворяли определённым условиям.
* **INDEX** — используется для быстрого создания данных базы данных.

Любое ограничение, которое вы определили, можно удалить с помощью команды **ALTER TABLE** с параметром **DROP CONSTRAINT**.

Некоторые реализации могут предоставлять ярлыки для отмены определённых ограничений.

Также некоторые реализации позволяют отключить ограничения. Вместо того, чтобы постоянно удалять ограничение из базы данных, можно временно отключить ограничение и включить их позже

## Базовые средства манипулирования данными в языке sql.

К базовым средствам манипулирования относятся UPDATE, DELETE и INSERT.

## Insert для вставки строк в существующие таблицы:

Общий синтаксис оператора *INSERT* выглядит следующим образом:

INSERT INTO table\_name

{ [ (column\_commalist) ] query\_expression

| DEFAULT VALUES

где query\_expression – может быть любым выражением-запросом.

Так например возможна вставкв всех строк какой-то таблицы:

INSERT INTO EMP (EMP\_NO, EMP\_NAME, EMP\_BDATE) TABLE EMP\_TEMP;

Вставка одной строки (используется конструктор строки):

INSERT INTO EMP

ROW (2445, 'Brown', '1985-04-08', 16500.00, 630, 772);

## Оператор update для модификации существующих строк в существующих таблицах

Общий синтаксис оператора UPDATE выглядит следующим образом:

UPDATE table\_name SET update\_assignment\_commalist

WHERE conditional\_expression

update\_assignment ::= column\_name =

{ value\_expression | DEFAULT | NULL }

Примеры:

Простой:

UPDATE EMP SET DEPT\_NO = 632, EMP\_SAL = EMP\_SAL + 1000.00

WHERE PRO\_NO = 772;

Сложный:

UPDATE EMP SET EMP\_SAL = (SELECT AVG (EMP1\_SAL)

FROM EMP EMP1

WHERE EMP.DEPT\_NO = EMP1.DEPT\_NO)

+ 1000.00, PRO\_NO = NULL

WHERE (SELECT EMP1.EMP\_SAL

FROM EMP EMP1, DEPT

WHERE EMP.DEPT\_NO = DEPT.DEPT\_NO

AND DEPT\_MNG = EMP1.EMP\_NO AND) > 30000.00;

## Оператор delete для удаления строк в существующих таблицах

Общий синтаксис оператора DELETE выглядит следующим образом:

DELETE FROM table\_name

WHERE conditional\_expression

Пример:

Простой:

DELETE FROM EMP WHERE PRO\_NO = 772;

Сложный:

DELETE FROM EMP WHERE EMP\_SAL >

(SELECT EMP1.EMP\_SAL

FROM EMP EMP1, DEPT

WHERE EMP.DEPT\_NO = DEPT.DEPT\_NO

AND DEPT.DEPT.MNG = EMP1.EMP\_NO);

* 1. **Понятие триггера. Механизм триггеров в SQL. Типы триггеров и их выполнение**.

**Триггером** называется хранимая в базе данных процедура, автоматически вызываемая СУБД при возникновении соответствующих условий.

В языке обеспечиваются возможности определения триггеров, которые вызываются («срабатывают») при вставке одной или нескольких строк в указанную таблицу, при модификации одной или нескольких строк в указанной таблице или при удалении одной или нескольких строк из указанной таблицы.

Таблица, с которой связывается определение триггера, называется *предметной таблицей* (subject table), а оператор SQL, выполнение которого приводит к срабатыванию триггера, мы будем называть *инициирующим* (triggering SQL statement).

Основными областями использования механизма триггеров являются:

* *Журнализация и аудит*. С помощью триггеров можно отслеживать изменения таблиц, для которых требуется поддержка повышенного уровня безопасности. Данные об изменении таблиц могут сохраняться в других таблицах и включать, например, идентификатор пользователя, от имени которого выполнялась операция обновления; временную метку операции обновления; сами обновляемые данные и т. д.
* *Согласование и очистка данных*. С любым простым оператором SQL, обновляющим некоторую таблицу, можно связать триггеры, производящие соответствующие обновления других таблиц. Например, с операцией вставки новой строки в таблицу EMP (прием на работу нового служащего) можно было связать триггер, модифицирующий значения столбцов DEPT\_EMP\_NO и DEPT\_TOTAL\_SAL162) строки таблицы DEPT со значением столбца DEPT\_NO, которое соответствует номеру отдела нового служащего.
* *Операции, не связанные с изменением базы данных*. В триггерах могут выполняться не только операции обновления базы данных. Стандарт SQL позволяет определять хранимые процедуры (которые могут вызываться из триггеров), посылающие электронную почту, печатающие документы и т. д.

Типы триггеров и их выполнение:

# **Триггеры before и after**

Если в определении триггера указано ключевое слово BEFORE, то триггер будет срабатывать непосредственно до выполнения операции обновления базовой таблицы соответствующим инициирующим оператором SQL. При задании ключевого слова AFTER триггер будет вызываться немедленно после выполнения инициирующего оператора.

## Триггеры insert, update и delete

Выбор одного из этих ключевых слов при определении триггера указывает на природу события, которое должно приводить к срабатыванию триггера. При задании ключевого слова INSERT к срабатыванию триггера может привести только выполнение операции вставки строк в предметную таблицу. Если указываются ключевые слова UPDATE или DELETE, то число возможных событий, приводящих к срабатыванию триггера, возрастает. Кроме явных операций модификации строк предметной таблицы или удаления из нее строк к срабатыванию триггера могут привести ссылочные действия

## Триггеры row и statement

Если в определении триггера присутствует конструкция FOR EACH ROW, то триггер будет вызываться для каждой строки предметной таблицы, обновляемой инициирующим SQL-оператором. Если же задано FOR EACH STATEMENT (или явная спецификация FOR EACH отсутствует), то триггер сработает один раз на всем протяжении процесса выполнения инициирующего SQL-оператора.

## Раздел when

Включение в определение триггера раздела WHEN с соответствующим условным выражением позволяет более точно специфицировать условие применимости триггера. Вычисление условного выражения производится над строками предметной таблицы, и триггер срабатывает только в том случае, когда значением условного выражения является true. Понятно, что виды и интерпретация логических выражений, допускаемых в разделе WHEN, различаются у триггеров с FOR EACH ROW и у триггеров с FOR EACH STATEMENT. В первом случае условное выражение вычисляется для одной строки, которая должна быть обновлена инициирующим SQL-оператором. Во втором – условное выражение вычисляется для всей предметной таблицы целиком и, по всей видимости, должно базироваться на «кванторных» предикатах. Следует также понимать, что вычисление условия раздела WHEN данного триггера производится только в том случае, если произошло событие срабатывания триггера.

**Выполнение триггеров.**

По ходу выполнения триггера возможна ситуация, при которой вызывается внутренний (вторичный) триггер, а также третичный и.т.д – глубина вложенности вызовов неограниченна. (Рекурсия, короче).

Выполнению триггера соответствует контекст выполнения. При завершении выполнения внутреннего триггера контекст меняется на контекст внешнего триггера.

## Общая структура оператора выборки в языке sql и схема выполнения.

Для выборки данных в прямом SQL используется оператор SELECT, возвращающий набориз одной или нескольких строк одинаковой структуры и задаваемый в следующем синтаксисе:

SELECT [ ALL | DISTINCT ] select\_item\_commalist

FROM table\_reference\_commalist

[ WHERE conditional\_expression ]

[ GROUP BY column\_name\_commalist ]

[ HAVING conditional\_expression ]

[ ORDER BY order\_item\_commalist ]

На первом шаге выполняется раздел FROM.

На втором шаге выполняется раздел WHERE. Условное выражение (conditional\_expression) этого раздела применяется к каждой строке таблицы T, и результатом является таблица T1, содержащая те и только те строки таблицы T, для которых результатом вычисления условного выражения является true. (Заголовки таблиц T и T1 совпадают.)

Если в операторе выборки присутствует раздел GROUP BY, то он выполняется на третьем шаге.

Каждый элемент списка имен столбцов (column\_name\_commalist), указываемого в этом разделе, должен быть одним из имен столбцов таблицы T1. В результате выполнения раздела GROUP BY образуется *сгруппированная таблица* T2, в которой строки таблицы T1 расставлены в минимальное число групп, таких, что во всех строках одной группы значения столбцов, указанных в списке имен столбцов раздела GROUP BY (*столбцов группировки*), одинаковы.

Если в операторе выборки присутствует раздел HAVING, то он выполняется на следующем шаге. Условное выражение этого раздела применяется к каждой группе строк таблицы T2, и результатом является сгруппированная таблица T3, содержащая те и только те группы строк таблицы T2, для которых результатом вычисления условного выражения является true. Условное выражение раздела HAVING строится по синтаксическим правилам, общим для всех условных выражений, но обладает той спецификой, что применяется к группам строк, а не к отдельным строкам. Поэтому предикаты, из которых строится это условное выражение, должны быть предикатами на группу в целом. В них могут использоваться имена столбцов группировки (*инварианты группы*) и так называемые агрегатные функции (COUNT, SUM, MIN, MAX, AVG) от других столбцов.

Выполнение раздела ORDER BY производится в последнюю очередь, следующим образом.Выбирается первый элемент списка сортировки, и строки таблицы T4 расставляются в порядке возрастания (если в элементе присутствует спецификация ASC; при отсутствии спецификации ASC/DESC предполагается наличие ASC) или в порядке убывания (при наличии спецификации DESC) в соответствии со значениями выражения, содержащегося в данном элементе, которые вычисляются для каждой строки таблицы T4. Далее выбирается второй элемент списка сортировки, и в соответствии со значениями заданного в нем выражения и порядка сортировки расставляются строки, которые после первого шага сортировки образовали группы с одинаковым значением выражения первого элемента списка сортировки. Операция продолжается до исчерпания списка элементов сортировки. Результирующий отсортированный список строк является окончательным результатом запроса.

# **54 Представляемые и порождаемые таблицы в sql. Агрегатные и кванторные функции.**

**Представляемая таблица**– представление таблице, которое можно использовать в операторе выборки наряду с базовыми таблицами

(table\_reference ::= table\_primary).

оператор создания представления в общем случае определяется следующими синтаксическими правилами (простой вид):

create\_view ::= CREATE VIEW table\_name

[ column\_name\_comma\_list ]

AS query\_expression - запрос

Имя таблицы, задаваемое в определении представления, существует в том же пространстве имен, что и имена базовых таблиц, и, следовательно, должно отличаться от всех имен таблиц (базовых и представляемых), созданных тем же пользователем.

Если имя представления встречается в *разделе FROM* какого-либо оператора выборки, то вычисляется *выражение запроса*, указанное в разделе AS, и оператор выборки работает с результирующей таблицей этого *выражения запроса*.

**Порождаемая таблица** derived\_table задается *выражением запроса*, заключенным в круглые скобки:

derived\_table ::= (query\_expression)

**Агрегатные и кванторные функции**

**агрегатные:**

Всё это функции над мультимножеством строк (сгруппированных, если GROUP BY, или всей таблицей). На осн. м/множества строк производятся множества значений

COUNT – количество значений

MAX

MIN

AVG – среднее значение

SUM

COUNT (\*) – количество строк (всегда считаются различными)

--------------------

(аргументы следующих **квантроных функций** – СУТЬ логические выражения)

EVERY: true , если лог. выр. True для всех строк

SOME (ANY): true , если лог. выр. True для хотя бы одной строки

DISTINCT – удаление дубликатов строк из результата.

Если на входе пустое множество, то

СOUNT = 0 EVERY = true MAX, MIN, AVG, SUM = NULL

## Предикаты языка sql.

**Логич. выражение** – булево выражение, которое строится на основе предикатов с исп.операторов AND, OR, NOT И ()

S – скал.произведение

R – строк.табл

|R| - значение в строке

T – таблица

Предикат позволяет специфицировать усл-е, результатом которого может быть true, false или unknown.

Для любых аргументов предикатов правила:

1. совместимости типов
2. равенство степеней строк. операндов |Rx| = |Ry|
3. для люб. предик. есть обратный NOT pred = NOT (pred)

**Предикаты:**

1. Предикат сравнения: Rx comp-op Ry
2. Предикат between: условие вх-я в диап. Значений (Rx between Ry and Rz)
3. Предикат is null: является неопр.значением аргумента
4. Предикат in: условие вхождения в указ.множество
5. Предикат like: string LIKE pattern ESCAPE symbol

в pattern - любой одиноч.символ, % - произв. п/строка

1. Предикат similar
2. Предикат exists: получили ли пустую таблицу (TRUE ⌠ | Трез |>0 , иначе FALSE)
3. Предикат unique: = TRUE, когда в Трезотсут. 2 одинак. стр
4. Предикат overlaps: пересечение по времени 2х событий
5. Предикат сравнения с квантором:

Rx comp\_op ALL T (квантор всеобщности)

Rx comp\_op SOME T (квантор существования)

1. Предикат match: соответствие подстроки рез-ту 2/л запросу
2. Предикат is distinct: Rx IS DISTINCT FROM Ry = TRUE ⌠ Rx и Ry – дубликаты

# **56. Поддержка авторизации доступа к данным в sql. Объекты и привилегии. Пользователи и роли.**

1) Метод авторизации в SQL относится к мандатным методам.

2) В SQL – 9 типов привилегий + привилегия на передачу привилегий.

3) Поддерживается принцип сокрытия информации об объектах от субъектов, у которых нет доступа.

4) Субъекты в SQL – пользователи и роли. Роль соответствует динамически определенной группе, каждая из которых отражает привилегии на исполнение данной роли, а также всеми привилегиями данной роли для доступа к объектам БД

## 57. Передача и аннулирование привилегий и ролей в sql.

1) Создатель любого объекта БД автоматически становится его владельцем. Владелец облает всеми привилегиями для выполнения действий над данным объектом. Он может передать все/часть своих привилегий другим пользователям или ролям, в том числе привилегию на передачу привилегий (рекурсия).

2) С любой пользой и любой ролью в SQL связан уникальный идентификатор инициализаций. Средства создания идентификатора пользователя в стандарте SQL не определено. Но должен существовать пользователь с идентификатором PUBLIC – псевдопольз., соотв. любому пользователю. Для создания/удаления роли поддерживаются операторы CREATEROLE и DROPROLE.

3) Для передачи привилегии роли поддерживается оператор GRANT, для удаления – REMOVE.

**58. Транзакции. Свойства ACID. Инициация и завершение транзакций в SQL. Проверка ограничений целостности и ее связь с механизмом транзакций. Точки сохранения.**

*Транзакция* - это последовательность операций, которые выполняются над базой данных. Система рассматривает эту последовательность как одну операцию и она обладает определенными свойствами, называемыми АСИД.

*Свойства АСИД:*

* Атомарность (Atomicity): транзакция выполняется как единая операция. Все операции внутри транзакции либо полностью отражаются в базе данных, либо не отражаются вовсе (это принцип "все или ничего").
* Согласованность (Consistency): транзакция переводит базу данных из одного согласованного состояния в другое. Транзакция считается успешно завершенной, только если действия операций не нарушают целостность базы данных.
* Изоляция (Isolation): две транзакции, которые выполняются одновременно, не должны влиять друг на друга. Результаты одной транзакции не должны быть видны другой до ее успешного завершения.
* Долговечность (Durability): после успешного завершения транзакции все изменения, внесенные ею, должны быть надежно сохранены в базе данных, даже в случае сбоев в аппаратуре или программном обеспечении.

**Инициация транзакций:**

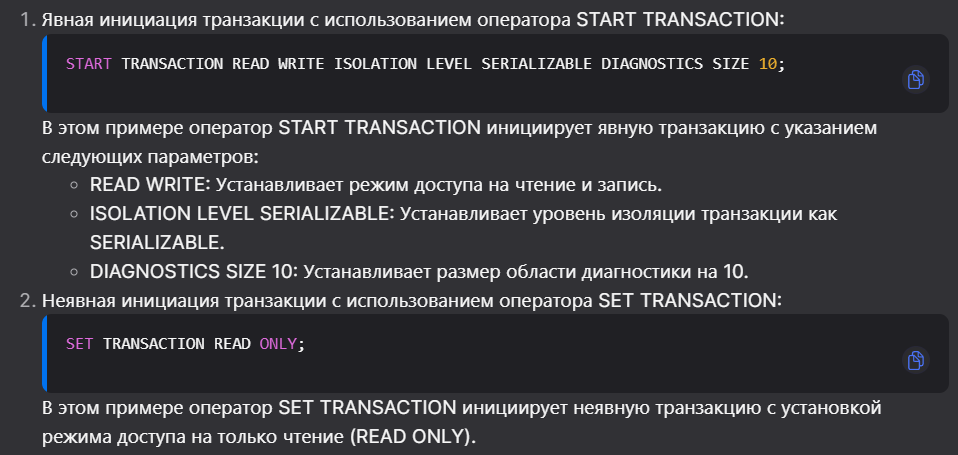
В SQL транзакции могут быть инициированы явно или неявно.

Явная инициация транзакции происходит с использованием оператора START TRANSACTION. При этом указывается режим доступа (READ ONLY или READ WRITE), уровень изоляции (READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ или SERIALIZABLE) и размер области диагностики (количество сохраняемых диагностических сообщений).

Большинство операторов SQL требуют наличие контекста транзакции, за исключением некоторых административных операторов.

Неявная инициация транзакции происходит, когда выполняется оператор, который требует контекста транзакции, но такого контекста не существует. В этом случае используются либо значения по умолчанию, либо значения, определенные оператором SET TRANSACTION.

Оператор SET TRANSACTION нельзя выполнять в контексте активной транзакции.

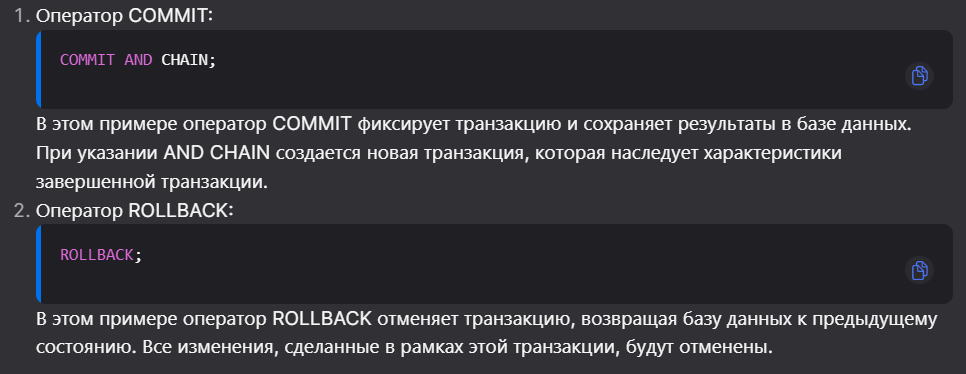


Завершение транзакций:

Чтобы завершить начатую транзакцию, пользователь должен явно выполнить один из двух операторов:

1. COMMIT [WORK] [AND [NO] CHAIN]: Этот оператор фиксирует транзакцию, то есть завершает ее и сохраняет результаты в базе данных. Если указано AND CHAIN, то после завершения текущей транзакции создается новая транзакция, которая наследует характеристики завершенной транзакции. Это позволяет экономить ресурсы, необходимые для создания новой транзакции.
2. ROLLBACK [WORK] [AND [NO] CHAIN]: Этот оператор отменяет транзакцию, возвращая базу данных к предыдущему состоянию. То есть все изменения, сделанные в рамках этой транзакции, будут отменены.

Оператор COMMIT считается успешно выполненным только после выполнения всех действий, необходимых для фиксации результатов транзакции. Это делается для защиты от сбоев, которые могут произойти во время выполнения COMMIT.



**Транзакции и ограничения целостности:**

В контексте выполняемой транзакции каждое ограничение целостности может находиться в одном из двух режимов:

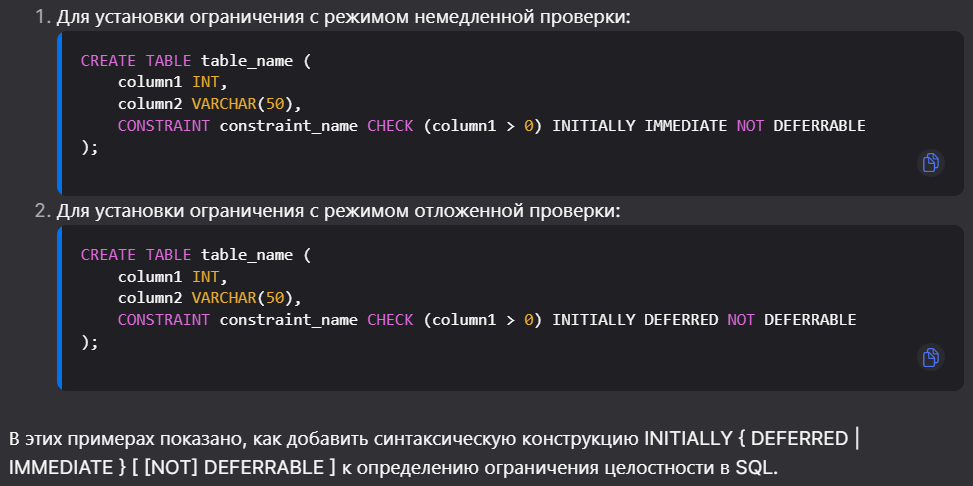
Режим немедленной проверки (immediate): Ограничение проверяется сразу после выполнения любой операции, которая изменяет состояние базы данных. Если операция нарушает хотя бы одно ограничение целостности в этом режиме, она будет отвергнута, но транзакция не будет отменена.

Режим отложенной проверки (deferred): Ограничение проверяется при выполнении операции COMMIT. В этом режиме допускается нарушение ограничений внутри транзакции, но все ограничения целостности должны быть соблюдены к моменту завершения транзакции. Если операции в транзакции нарушают хотя бы одно отложенно проверяемое ограничение целостности, транзакция будет отменена (COMMIT будет рассматриваться как ROLLBACK).

Для указания режима проверки к определению ограничения целостности добавляется следующая синтаксическая конструкция:

INITIALLY { DEFERRED | IMMEDIATE } [ [NOT] DEFERRABLE ]

По умолчанию предполагается INITIALLY IMMEDIATE NOT DEFERRABLE. Комбинация INITIALLY DEFERRED NOT DEFERRABLE недопустима.



Точки сохранения в транзакциях:

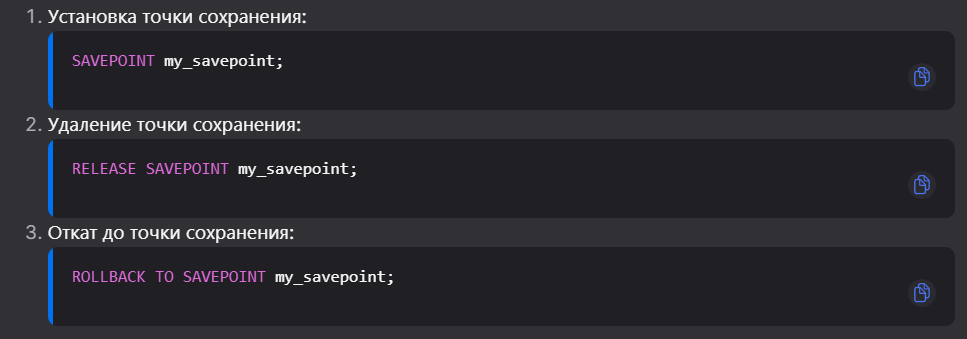
Точка сохранения - это метка в последовательности операций транзакции, которую можно использовать для частичного отката транзакции с сохранением результатов операций, выполненных до этой метки.

Вот некоторые операторы, связанные с точками сохранения:

* Установка точки сохранения: SAVEPOINT savepoint\_name
* Удаление точки сохранения: RELEASE SAVEPOINT savepoint\_name
* Откат до точки сохранения: ROLLBACK TO SAVEPOINT savepoint\_name

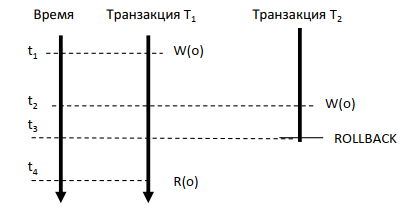
В одной транзакции можно установить несколько последовательных точек сохранения. При откате до определенной точки сохранения автоматически выполняется RELEASE для всех точек, определенных в транзакции после этой метки.

Точки сохранения не нарушают принцип атомарности транзакции, потому что извне транзакции она все равно рассматривается как единая операция (точки сохранения невидимы для внешнего пользователя).



**59. Сериализация транзакций: виды конфликтов транзакций и порождаемые ими феномены поведения транзакций. Уровни изоляции транзакций в SQL.**

**Феномен "Потерянные изменения"**:

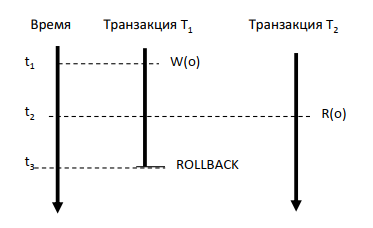


Ситуация "потерянных изменений" нарушает требование изолированности транзакций. Поэтому, чтобы обеспечить изолированность одновременно выполняемых транзакций, СУБД должна предотвращать потерю изменений.

Чтобы избежать такой ситуации, в транзакции T1 требуется заблокировать доступ к объекту o для изменений до завершения транзакции T1. Таким образом, другая транзакция (например, T2) не сможет изменять этот объект до завершения T1.

Феномен "потерянных изменений" возникает, когда транзакция T2 изменяет объект o базы данных, который уже был изменен транзакцией T1. Затем, если транзакция T2 завершается аварийно (например, из-за нарушения ограничений целостности) или при вызове ROLLBACK, то продолжающая выполняться транзакция T1 не увидит своих изменений объекта o, которые она сделала до этого.

**Феномен "Грязное чтение":**



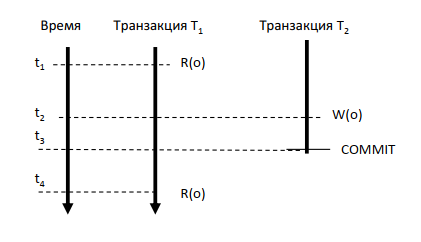
Эта ситуация также нарушает требование изолированности пользователей, поскольку каждый пользователь должен начинать свою транзакцию с согласованного состояния базы данных и имеет право видеть только согласованные данные.

Чтобы избежать ситуации "грязного чтения" данных, необходимо заблокировать доступ к чтению объекта o до завершения транзакции T1, которая его изменяет. Таким образом, другая транзакция (например, T2) не сможет читать этот объект до завершения T1.

Феномен "грязного чтения" возникает, когда транзакция T2 читает объект o базы данных, который уже был изменен транзакцией T1. Поскольку транзакция T1 еще не завершена, транзакция T2 видит несогласованные ("грязные") данные.

Впоследствии, если транзакция T1 завершается аварийно, это может привести к появлению "грязных" данных в транзакции T2.

**Феномен "Неповторяющееся чтение":**

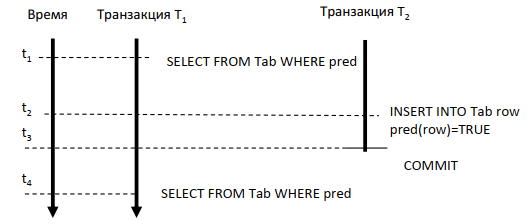


Чтобы избежать неповторяющихся чтений, никакая другая транзакция не должна изменять объект o до завершения транзакции T1. Для этого достаточно заблокировать доступ к записи объекта o до завершения T1.

Это обычно является основным требованием для обеспечения изолированности транзакций, хотя отсутствие неповторяющихся чтений само по себе не гарантирует полной изолированности пользователей.

Феномен "неповторяющегося чтения" возникает, когда транзакция T2 изменяет объект o базы данных, который уже был прочитан транзакцией T1, и фиксирует свои изменения. Затем, если транзакция T1 повторно читает тот же самый объект после фиксации изменений транзакцией T2, то транзакция T1 видит измененное состояние объекта, которое отличается от состояния при первом чтении.

**Феномен "Фантомы":**



Чтобы избежать появления фантомных строк, требуется использовать более высокий уровень изоляции транзакций.

Идеи для такого механизма (например, предикатные синхронизационные блокировки) были предложены еще во время разработки проекта System R, но в большинстве систем они не реализованы. Тем не менее, ситуация с появлением фантомных строк противоречит идее изолированности транзакций.

Пример ситуации фантомов: Транзакция T1 выбирает данные из таблицы Tab на основе определенного условия. Перед завершением T1 транзакция T2 добавляет в таблицу Tab строку, которая соответствует тому же условию, и фиксирует свои изменения. Затем, при повторной выборке из таблицы Tab с тем же условием в транзакции T1, результат будет содержать дополнительную "фантомную" строку.

**уровни изоляции:**



60. Сериализация транзакций: сериальный план. Основные подходы к сериализации. Двухфазный протокол синхронизационных блокировок.

*Сериализация транзакций* - это способ управления выполнением нескольких операций в базе данных, чтобы обеспечить их изолированность. Представьте, что в базе данных одновременно выполняется несколько операций. Чтобы результаты этих операций были правильными и надежными, нужно следить за тем, как они выполняются.

*Сериальный план* - это способ выполнения операций, при котором они выполняются последовательно или параллельно, но так, чтобы результат был таким же, как если бы они выполнялись по очереди. Система управления базами данных (СУБД) отвечает за обеспечение такого способа выполнения.

*Сериализация транзакций* - это механизм, который гарантирует, что операции выполняются по правильному плану. Это важная функция СУБД и ее компонента, называемого менеджером транзакций. Система с поддержкой сериализации транзакций обеспечивает надежность и изолированность для пользователей базы данных.

Основная проблема заключается в выборе метода сериализации, который не ограничивает слишком сильно чередование операций или реальную параллельность. То есть, нужно найти баланс между эффективностью и правильностью выполнения операций.

Когда мы говорим о последовательном выполнении транзакций, это может показаться самым простым решением. Но есть ситуации, когда можно выполнять операции разных транзакций в любом порядке и все равно сохранять порядок их результатов. Например, это может быть применимо к транзакциям только для чтения или к транзакциям, которые не конфликтуют между собой по данным в базе данных.

Между двумя транзакциями T1 и T2 могут возникать следующие виды конфликтов:

* W/W (запись/запись): транзакция T2 пытается изменить объект, который еще не закончила изменять транзакция T1 (это может привести к потере изменений);
* W/R (запись/чтение): транзакция T2 пытается прочитать объект, который еще не закончила изменять транзакция T1 (это может привести к "грязному" чтению, когда данные могут быть некорректными);
* R/W (чтение/запись): транзакция T2 пытается изменить объект, который уже прочитала незавершенная транзакция T1 (это может привести к неповторимому чтению, когда данные могут быть несогласованными).

Практические методы сериализации транзакций основываются на учете этих конфликтов и выборе правильного порядка выполнения операций.

**Существуют два основных подхода к сериализации транзакций**, которые помогают управлять конфликтами и обеспечивать правильный порядок выполнения операций в базе данных.

*Первый подход* основан на синхронизационных захватах объектов базы данных. Это означает, что каждая транзакция должна получить доступ к объекту базы данных, прежде чем ее можно будет выполнить. Если другая транзакция уже заблокировала этот объект, то текущая транзакция должна ждать, пока объект не станет доступным. Этот подход называется пессимистическим, потому что он предполагает, что конфликты будут возникать часто, и они должны быть разрешены немедленно.

*Второй подход* основан на использовании временных меток. Каждая транзакция получает временную метку при начале выполнения. Результаты всех операций модификации базы данных сохраняются в рабочей памяти транзакции. Фактическая модификация базы данных и разрешение конфликтов происходят только на стадии фиксации транзакции. Этот подход называется оптимистическим, потому что он предполагает, что конфликты будут редкими, и разрешение происходит только при фиксации.

В данном случае мы ограничимся рассмотрением более распространенных пессимистических методов сериализации транзакций.

В централизованных базах данных, особенно в системах клиент-серверной архитектуры, наиболее распространенным подходом является использование синхронизационных блокировок. Один из таких подходов называется **двухфазным протоколом синхронизационных захватов** объектов баз данных (Two-Phase Locking Protocol, 2PL).

Вкратце, этот подход заключается в следующем: перед выполнением любой операции внутри транзакции T над определенным объектом базы данных o, транзакция T запрашивает синхронизационную блокировку для объекта o в определенном режиме, в зависимости от типа операции.

Режим совместного доступа (S - Shared) означает, что объект блокируется для совместного чтения и требуется для операций чтения объекта.

Режим монопольного доступа (X - Exclusive) означает, что объект блокируется для монопольной записи и требуется для операций вставки, удаления и изменения объекта.

Это позволяет контролировать доступ к объектам базы данных и предотвращать конфликты при одновременном доступе нескольких транзакций к одному и тому же объекту.

**61. Сериализация транзакций: гранулированные и предикатные блокировки.**

**Гранулированные синхронизационные блокировки:**

Файлы, таблицы и кортежи. Уровень объекта определяется операцией, которую мы хотим выполнить. Например, для удаления таблицы нам нужна блокировка всей таблицы, а для удаления кортежа - только этого кортежа.

Объект любого уровня может быть заблокирован в режиме S (совместный доступ) или X (исключительный доступ).

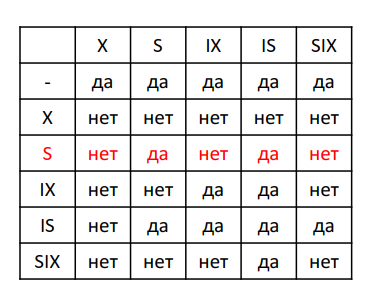
Чтобы согласовать блокировки разных уровней, мы используем протокол *гранулированных блокировок* и вводим новые типы блокировок:

Intented for Shared lock (IS): означает намерение заблокировать объект o' внутри объекта o в режиме S. Например, если мы читаем кортежи из таблицы Tab, мы должны заблокировать таблицу Tab в режиме IS, а до этого заблокировать файл, где находится таблица Tab, также в режиме IS.

Intented for eXclusive lock (IX): означает намерение заблокировать объект o' внутри объекта o в режиме X. Например, для удаления кортежей из таблицы Tab мы должны заблокировать таблицу Tab в режиме IX, а до этого заблокировать файл, где находится таблица Tab, также в режиме IX.

Shared, Intented for eXclusive lock (SIX): означает совместную блокировку S всего составного объекта с намерением впоследствии заблокировать внутренние объекты в режиме X. Например, если мы выполняем длительную операцию просмотра таблицы Tab с возможностью удаления некоторых просматриваемых кортежей, то наиболее эффективно будет заблокировать таблицу Tab в режиме SIX, а до этого заблокировать файл, где находится таблица Tab, в режиме IS.

Таким образом, гранулированные синхронизационные блокировки позволяют нам эффективно управлять доступом к объектам базы данных на разных уровнях.



**Предикатные синхронизационные блокировки** - это способ решения проблемы фантомов в базах данных. Вместо блокировки отдельных объектов, таких как таблицы или кортежи, блокировка основывается на условиях или предикатах, которым должны удовлетворять объекты.

Например, при выполнении операций над базой данных, которые задаются условием, мы можем требовать синхронизационную блокировку для этого условия. Это позволяет избежать проблемы фантомов и обеспечивает сериализацию транзакций.

Однако использование предикатных блокировок может быть сложным, так как условия, допустимые в языке SQL, могут быть разнообразными. Поэтому требуется определить совместимость двух предикатных блокировок.

В проекте System R предлагается использовать предикатные блокировки для решения проблемы фантомов. При сканировании таблицы по индексу блокируется не только сам кортеж, но и его идентификатор (tid). Также предлагается блокировать пару "идентификатор индекса, интервал значений ключа этого индекса". Две такие блокировки считаются совместимыми, если они не пересекаются или совместимы согласно таблице совместимости.

Однако у этого подхода есть недостатки. Например, при сканировании таблиц без использования индексов, можно гарантировать отсутствие фантомов только при блокировке всей таблицы. Также условие выборки кортежа может быть строже указанного диапазона сканирования, что может привести к сильной блокировке и затронуть больше кортежей, чем требуется для выполнения операции сканирования.

Предикатные синхронизационные блокировки простых условий - это более продвинутый подход к решению проблемы фантомов в базах данных. В этом подходе используются простые условия, которые представляют собой простые предикаты сравнения, например "столбец = значение".

В типичных базах данных только простые условия могут быть использованы в интерфейсе управления памятью. Сложные условия в операторе SQL компилируются в последовательность обращений к управлению памятью, каждое из которых содержит только простые условия.

Простое условие явно указывается при открытии сканирования таблицы, либо напрямую, либо через индекс. Также можно указать цель сканирования: выборка, удаление или обновление кортежей.

Неявные условия задаются операциями вставки, удаления и обновления кортежей. Например, все столбцы таблицы могут быть заданы в конъюнктивном логическом выражении вида "столбец = значение".

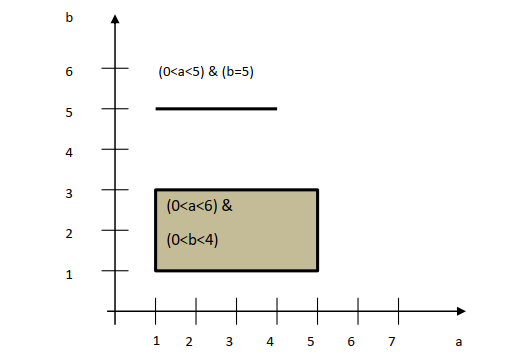
В случае SQL-ориентированных баз данных простые условия могут быть использованы для предикатных блокировок.

Совместимость предикатных блокировок простых условий определяется на основе геометрической интерпретации. Каждое простое условие можно представить как k-мерный прямоугольник в n-мерном пространстве значений таблицы. Два простых условия считаются совместимыми, если их прямоугольники не пересекаются.

Давайте рассмотрим следующую ситуацию: у нас есть два простых условия, scond1 и scond2. Предположим, что транзакция T1 запрашивает блокировку для scond1, а транзакция T2 - для scond2, в режимах, которые обычно были бы несовместимы для базы данных. Однако, эти блокировки могут быть совместимыми только в том случае, если прямоугольники, соответствующие scond1 и scond2, не пересекаются.

Это утверждение легко понять: каждому прямоугольнику в n-мерном пространстве возможных значений таблицы соответствует некоторое подмножество возможных значений. Отсутствие пересечения у двух прямоугольников гарантирует отсутствие конфликтов между транзакциями.

Чтобы проиллюстрировать это, рассмотрим пример: допустим, что транзакция T1 требует блокировку для условия (0 < a < 5) & (b = 5), а транзакция T2 требует блокировку для условия (0 < a < 6) & (0 < b < 4). Вне зависимости от того, в каких режимах эти транзакции запрашивают блокировки, эти блокировки всегда будут совместимыми, так как прямоугольники, соответствующие условиям, не пересекаются.

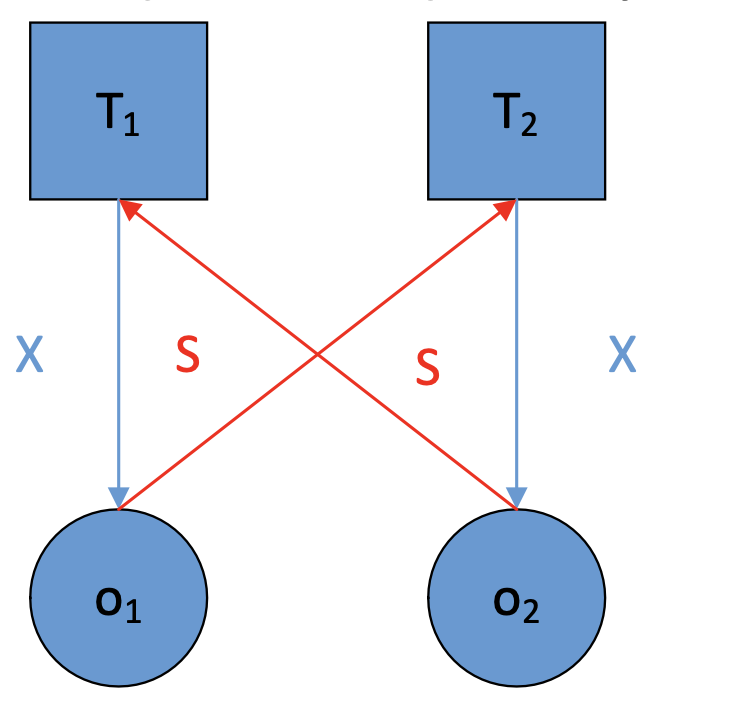


С помощью системы блокировок простых условий можно избежать использования детализированных блокировок:

Чтобы заблокировать всю таблицу, достаточно заблокировать условие, которое охватывает все строки (например, &1 ≤i ≤n, где min(mi) < ai < max(mi)).

Чтобы заблокировать всю базу данных, достаточно заблокировать условие, которое является комбинацией условий блокировки всех таблиц в этой базе данных.

Блокировки простых условий описываются таблицами, которые не сильно отличаются от таблиц в традиционных синхронизаторах с детализированными блокировками. Поэтому внедрение механизма предикатных блокировок в базу данных не вызывает значительных сложностей.

62. Синхронизационные тупики, способы их обнаружения и разрушения.

Одним из самых больших недостатков метода сериализации транзакции на основе синхронизационных блокировок является возможность возникновения тупиков между транзакциями. Тупик — это ситуация, когда (в самом простом случае) есть две транзакции, желающие получить доступ в несовместимых режимах к двум объектам, и каждая получила доступ к одному. При попытке получить доступ к второму объекту они вынуждены ждать друг друга, то есть никогда не прийти к успеху.

Основой обнаружения тупиков является построения графа ожидания транзакций — ориентированного двудольного графа, в котором одним типом вершин являются транзакции, другим — объекты блокировки. Дуги соединяют только разнотипные вершины. Дуга из транзакции к объекту значит, что транзакция заблокировала объект. Дуга из объекта в транзакцию значит, что транзакция хочет заблокировать объект. Очевидно, что если есть цикл, то есть тупик. Для распознавания тупиков периодически строится граф и производится поиск тупиков. Традиционной техников поиска циклов в ориентированном графе является редукция графа. Удаляются все дуги, исходящие из транзакций, в которые не входят дуги — эти транзакции уже получили все блокировки и не могут оказаться в тупике. Удаляются дуги, входящие в транзакции, из которых не исходит дуг — эти транзакции еще ничего не блокируют, поэтому не могут стать частью тупика. Для тех объектов, для которых не осталось входящих дуг, но существуют исходящие, ориентация одной из них изменяется на противоположную (это моделирует удовлетворение запроса блокировки), и действия повторяются, пока граф не перестанет меняться. Если в графе останутся дуги, то они образуют цикл.

Разрушение тупика начинается с выбора в цикле транзакций транзакции-жертвы — той транзакции, которой решено пожертвовать, чтобы обеспечить возможность продолжения работы других транзакций. Выбрать жертву сложно, так как существует много противоречивых критериев. С одной стороны, убив транзакцию, удерживающую наибольшее число блокировок объектов, мы освободим много объектов, что с большой вероятностью приведет к решению тупика, с другой стороны на убитую транзакцию было потрачено много системных ресурсов, завершать ее неразумно с системной точки зрения. Можно убивать самую молодую транзакцию, которая израсходовала мало ресурсов, но одновременно у нее и мало блокировок, так что она с большой вероятностью не решит тупик. Можно выбирать транзакцию случайным образом, что в среднем даст хороший результат, но здесь не учитывается возможная приоритетность транзакций.

После выбора жертвы ее откатывают (полностью или частично). Это нарушение принципа изолированности пользователей, но избежать этого нельзя.

63. Сериализация транзакций на основе временных меток.

Основной идеей метода временных меток является следующее: если транзакция T1 началась раньше T2, то система обеспечивает такой сериальный план, как если бы T1 была полностью выполнена до начала T2. Для этого каждой транзакции T предписывается временная метка t(T), соответствующая времени начала выполнения транзакции T. При выполнении операции над объектом транзакция помечает его своим идентификатором, временной меткой и типом операции (чтение или изменение). Перед выполнением операции над объектом транзакция T2 выполняет следующие действия: проверяет, помечен ли объект. Если нет, метит и продолжает работать. Иначе проверяет, завершилась ли пометившая транзакция. Если завершилась, помечает и продолжает работать. Иначе проверяет конфликтность операций — если операции совместимы, то запоминается идентификатор и время T2 и транзакция продолжает работать. Если операции конфликтуют, то если пометившая объект транзакция моложе, то производится откат ее и всех других транзакций, идентификаторы которых сохранены при объекте, и T2 продолжает работать. Если пометившая объект транзакция старше, то производится откат транзакции T2, T2 получает новую временную метку и начинает заново.

К недостаткам этого метода относят потенциально более частые откаты транзакций, чем при использовании синхронизационных захватов, что связано с более грубым определением конфликтности транзакций. Кроме того, в распределенных системах сложно работать с временными метками, чтобы они были корректны. Но зато не нужно распознавать тупики, а построение графа ожиданий в распределенных системах очень дорогое.

64. Версионный вариант алгоритма временных меток.

Основной идеей версионных алгоритмов является допущение существования в базе данных нескольких версий одного и того же объекта, что позволяет, главным образом, выполнять операции чтения над некоторой предыдущей версией объекта базы данных. В результате чтение выполняется без задержек и тупиков, а также без некоторых откатов.

В алгоритме MVTO (Multiversion Timestamp Ordering) порядок выполнения операций одновременно выполняемых транзакций задается порядком временных меток, которые транзакции получают в момент старта. Временные метки также используются для идентификации версий данных при чтении и модификации — каждая версия получает временную метку той транзакции, которая ее записала. Алгоритм не только следит за порядком выполнения операций транзакций, но и отвечает за трансформацию операций над объектами в операции над версиями объектов.

Временная метка, полученная транзакцией T в начале работы, будет обозначаться t(T), операция чтения объекта в транзакции Ti – Ri(O), для обозначения чтения i-й транзакции версии, созданной k-й транзакции будем использовать Ri(Ok), запись транзакцией Ti версии объекта — Wi(Oi).

Алгоритм работает следующим образом:  
Ri(O) → Ri(Ok), где Ok – версия объекта, помеченная наибольшей временной меткой t(Tk) такой, что t(Tk) <not> = t(Ti), то есть версия, созданная транзакцией, которая не моложе Ti, но самая молодая из создававших версии объекта.  
Wi(O): если есть незафиксированная Tn, выполнившая Rn(Ok): t(Tk) <not> = t(Ti) <not> t(Tn), то запись не выполняется, Ti откатывается. Иначе Wi(O) → Wi(Oi), то есть образуется еще одна версия объекта.

При откате любой транзакции уничтожаются все созданные ею версии объектов базы данных и откатываются все транзакции, прочитавшие хотя бы одну из этих версий. То есть откаты транзакций могут быть каскадными.

Выполнение операции фиксации откладывается, пока не завершатся все транзакции, записавшие версию данных, прочитанную Ti. Без этого условия не соблюдалось бы свойство долговечности, так как пришлось бы откатывать успешно завершившиеся транзакции.

Основным преимуществом данного метода является отсутствие задержек и откатов при выполнении чтения, а недостатков — возможность возникновения каскадных откатов при записи и накопление версий одного и того же объекта, для которых сложно определить, какие из них больше не нужны.

65. Версионный вариант двухфазного протокола синхронизационных блокировок.

В версионном варианте двухфазного протокола синхронизационных блокировок поддерживается до двух версий объектов базы данных. Текущей версией объекта называется версия, созданная зафиксированной транзакцией с наиболее поздним временем фиксации. Незафиксированной версией — версию, созданную еще не завершившейся транзакцией. В каждый момент времени может существовать не более одной незафиксированной версии каждого объекта.

Операции выполняются следующим образом: чтение ведется из текущей версии объекта. Запись, которая приводит к созданию новой версии объекта, выполняется только после завершения (фиксации или отката) транзакции, создавшей незафиксированную версию объекта. Выполнение фиксации транзакции откладывается до тех пор, пока не завершатся все транзакции, читавшие текущие версии объектов, которые завершающаяся транзакция должна заменить незафиксированными версиями.

Для реализации такого поведения используются специальные виды синхронизационных блокировок: ReadLock – блокировка объекта перед чтением его текущей версии, гарантирует, что при повторном чтении будет прочитано то же самое. WriteLock – любой объект блокируется в этом режиме перед операцией, приводящей к созданию новой версии объекта, гарантирует, что в любой момент времени будет существовать не более одной незафиксированной версии объекта. CommitLock – блокировка на время фиксации транзакции, затрагивает все объекты, чью новую версию эта транзакция создавала. Удовлетворение этой блокировки гарантирует, что все транзакции, которые читали объекты, которые были изменены текущей транзакцией, завершились.

Совместимы RL/RL (очевидным образом), RL/WL и WL/RL (за счет существования двух версий, то есть сохранения версии для чтения). Тупики возможны. (Например, если две транзакции хотят изменить два объекта, и каждая из них успела захватить по одному объекту, они обе должны ждать, пока другая не завершиться).

66. Версионно-блокировочный протокол сериализации транзакций для поддержки только читающих транзакций.

Только читающая транзакция при создании получает временную метку. При чтении объекта транзакция получает доступ к его версии, которая была образована транзакцией, хронологически последней зафиксировавшейся до начала текущей транзакции.

Плюсом данного протокола по отношению с версионным вариантом двухфазного протокола является полное отсутствие синхронизационных задержек при работе только читающих транзакций. К тому же эти транзакции никогда не откатываются. Минусом является появление в базе данных произвольного числа версий объектов (вместо двух у двухфазного протокола). Поэтому нужен сборщик мусора, который будет удалять ненужные версии данных. Простейший сборщик удаляет все неиспользуемые версии, временные метки которых меньше временной метки самой старой из живых читающих транзакций.

67. Ситуации, требующие восстановления базы данных и общие принципы восстановления информации. Понятие журнала и основные походы к его ведению. Индивидуальные откаты транзакций.

**Ситуации, при которых требуется производить восстановление состояния БД**

1. Индивидуальный откат транзакции (в случае завершения оператором ROLLBACK явного или системой) [нарушение отложенных ограничений целостности, выбор транзакции в качестве «жертвы» при разрушении тупика, возникновение исключительной ситуации в прикладной программе] [нужно устранить последствия операторов модификации базы данных, которые выполнялись в этой транзакции]

2. Восстановление после внезапной потери содержимого оперативной памяти (мягкий сбой) [аварийное выключения электропитания, при возникновения неустранимого сбоя процессора, срабатывания контроля основной памяти] [потеря части базы данных, которая к моменту сбоя содержалась в буферах оперативной памяти СУБД]

3. Восстановление после поломки внешней памяти (жесткий сбой) [поломка основного внешнего носителя базы данных] [основой восстановления является архивная копия и журнал изменений базы данных.]

Общими принципами восстановления являются следующие:

результаты зафиксированных транзакций должны быть сохранены в восстановленном состоянии базы данных, а незафиксированных должны отсутствовать

восстанавливается последнее по времени согласованное состояние базы данных.

Журнал – в нём хранятся избыточные данные, содержащие последовательность записей об изменении базы данных.

**основные варианты ведения журнальной информации:**

1. Поддержка отдельных локальных журналов изменений БД для каждой транзакции в основной памяти СУБД (используются для индивидуальных откатов транзакций) и общего журнала изменений БД во внешней памяти (используется для восстановления состояния БД после мягких и жестких сбоев). [быстро выполнять индивидуальные откаты транзакций, но приводит к дублированию информации в локальных и общем журналах]

2. Поддержка только общего журнала изменений БД, который используется и при выполнении индивидуальных откатов. [чаще используется]

**Индивидуальный откат транзакции:**

* все записи в журнале от данной транзакции связываются в обратном хронологическом порядке [в списке].
* В начале списка для незавершенных транзакций находится запись о последнем изменении базы данных (могут находиться в буфере основной памяти), а для закончившихся транзакций, является запись о конце транзакции (уже вытолкнуты в пзу)
* Концом списка всегда служит первая запись об изменении базы данных, всегда можно восстановить прямой список записей из за наличия id.

**Индивидуальный откат транзакции (это возможно только для незавершенных транзакций) выполняется следующим образом:**

1. 1. Выбирается очередная журнальная запись из списка данной транзакции;
2. 2. Выполняется противоположная по смыслу операция, восстанавливающая предыдущее состояние объекта базы данных и также она журнализуется
3. 3. При успешном завершении отката в журнал заносится запись о конце транзакции. С точки зрения журнала такая транзакция является зафиксированной.

68.  **Управление буферным пулом основной памяти. Буферизация (кеширование) блоков базы данных и журнала.**

Управление буферным пулом БД происходит если при выполнении некоторой операции в некоторой транзакции требуется доступ к некоторому блоку базы данных, и копия этого блока отсутствует в буферном пуле,

СУБД должна (у неё собственная стратегия замещения страниц буферного пула):

• выделить какую-либо страницу буферного пула,

• считать в нее с диска требуемый блок базы данных,

• предоставить доступ к этой странице запросившей операции,

• если в буферном пуле нет свободных страниц, тогда СУБД в соответствии с некоторым критерием выбора находит некоторую занятую страницу (для “откачки”) и освобождает (выталкивает в пзу).

Критерий - алгоритма LRU с приоритетами страниц. Так как СУБД располагает большей информацией о страницах буферного пула [подсистема управления буферным пулом «знает», что страница может быть использована, какой блок базы данных потребуется после завершения просмотра («упреждающая выборка»), некоторые блоки базы данных заведомо требуются чаще других блоков]

в стратегии замещения страниц буферного пула базы данных используется алгоритм LRU с приоритетами страниц:

• Высокоприоритетные страницы стареют (т.е. становятся кандидатами на замещение) медленнее, чем низкоприоритетные страницы.

• страницы, содержащие копии корневых блоков индексов, являются настолько высокоприоритетными, что обычно никогда не замещаются.

• поддерживается предварительное считывание в буферную память копий блоков, доступ к которым вскоре понадобится.

**Буферизация (кеширование) блоков базы данных и журнала.**

Журнализация операций изменения базы данных (операций уровня ядра СУБД) связана с управлением транзакциями, и с буферизацией (кешированием) блоков базы данных в основной памяти.

Буферизация блоков базы данных в основной памяти является тем с помощью чего можно получить основной прирост к скорости работы. Аналогично для журнала записи в журнал тоже буферизуются а не немедленно перемещалась во пзу.

Для буферизации записей журнала обычно используются два буфера для ускорения работы находящиеся в физической основной памяти управляемой непосредственно СУБД а не виртуальной управляемой ОС [может удалить буферную страницу СУБД из основной памяти и перенести ее копию во внешнюю память в область свопинга, а после обратно при попытки записи вернет . “неожиданный обмен”] (техника “push-pull”). После полного заполнения первый буфер выталкивается на магнитный диск, и пока совершается этот обмен, журнальные записи размещаются во втором буфере. К моменту конца обмена заполняется второй буфер, он выталкивается во внешнюю память, а журнальные записи снова размещаются в первом буфере и т.д.

69. **Физическая синхронизация. Протокол 2PL для физической синхронизации и борьба с физическими синхронизационными тупиками.**

**Физическая синхронизация**

Координацию параллельного доступа к страницам буферного пула (в случае одновременного выполнения тразакций) не обеспечивает логическая синхронизация (на уровне кортежей, таблиц, файлов), используемая для сериализации транзакций.

логическая синхронизация может легко допустить параллельное выполнение нескольких операций, требующих обновления одного и того же индекса. Некоординированное параллельное обновление B-дерева с большой вероятностью приводит к разрушению его структуры

При выполнении операций низкого уровня (ядра СУБД) необходимо поддерживать дополнительную «физическую» синхронизацию, в которой единицами блокировки служат страницы буферного пула (или блоки) базы данных.

В пределах операции перед чтением из страницы буферного пула блока базы данных требуется запросить у подсистемы управления буферным пулом блокировку соответствующей страницы блока в режиме S, а перед записью в страницу (в блок) – ее блокировку в режиме X.

Совместимость блокировок как для транзакций. Блокировки страниц буферного пула нужны для координации параллельного доступа к страницам при параллельном выполнении транзакций а также при возникновении ошибки при выполнении операций низкого уровня в середине операции)

Достаточным условием (хотя и слишком жестким) корректного выполнения операций является соблюдение **двухфазного протокола синхронизационных блокировок** над страницами буферного пула в пределах операций.

Это условие не является необходимым. Каждую операцию уровня ядра СУБД можно разбить на последовательность «микроопераций» и потребовать соблюдения двухфазного протокола синхронизационных блокировок в пределах микроопераций.

Общий принцип состоит в том, что в пределах одной микрооперации блокируются все блоки базы данных, которые обязаны быть изменены согласованным образом.

**Суть протокола в том что для** борьбы с физическими синхронизационными тупиками можно обойтись без построения и анализа графа ожидания. Число разных микроопераций в ядре ограничено и не слишком велико. Для каждой микрооперации известно максимальное число физических блокировок (и операций записи/чтения блоков базы данных), которые могут потребоваться при ее выполнении. Если известно максимальное число транзакций, которые могут одновременно выполняться в СУБД, то можно оценить максимальное время ожидания доступа к буферной странице, если отсутствует синхронизационный тупик.

**Тогда если** при выполнении микрооперации транзакция не дожидается разрешения доступа к запрошенному блоку базы данных за установленное время, она откатывает недовыполненную микрооперацию с восстановлением состояния измененных буферных страниц, снимает все установленные физические блокировки и выполняет микрооперацию заново.

70. **Протокол Write Ahead Log и его связь с буферизацией.**

Имеются два вида буферов которые выталкивается в пзу:

• буферы журнала [его полное заполнение журнальными записями]

• буферный пул страниц основной памяти [переместить в основную память некоторый блок базы данных, а свободных страниц в буферном пуле нет. срабатывает алгоритм замещения страниц]

если произошел мягкий сбой (содержимое буферов утрачено), то для проведения восстановления базы данных необходимо иметь некоторое согласованное состояние журнала и базы данных во внешней памяти.

Протокол упреждающей записи в журнал (Write Ahead Log) - протокол журнализации (и управления буферизацией) состоит в том, что если требуется вытолкнуть во внешнюю память буферную страницу, содержащую измененный объект базы данных, то перед этим нужно гарантировать выталкивание во внешнюю память журнала буферной страницы журнала, содержащей запись об изменении этого объекта.

Основной принцип протокола - запись об изменении объекта базы данных должна оказаться во внешней памяти журнала раньше, чем сам измененный объект окажется во внешней памяти базы данных.

При следовании протоколу WAL:

• если во внешней памяти базы данных находится некоторый объект базы данных, по отношению к которому выполнена операция модификации, то во внешней памяти журнала обязательно находится запись, соответствующая этой операции. Но обратное неверно

• Каждая успешно завершенная транзакция должна быть реально зафиксирована во внешней памяти и какой бы сбой не произошел система может восстановить состояние базы данных до момента сбоя.

минимальным требованием, гарантирующим возможность восстановления последнего согласованного состояния базы данных, является: выталкивание при фиксации транзакции во внешнюю память журнала всех записей об изменении базы данных этой транзакцией; при этом последней записью в журнал, производимой от имени данной транзакции, является специальная запись о конце транзакции.

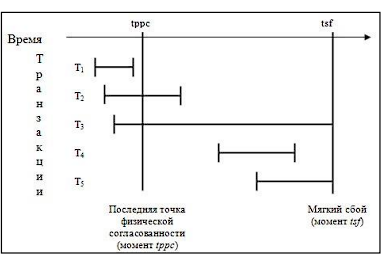
71. Ф**изически согласованное состояние базы данных. Схема восстановления базы данных после мягкого сбоя до точки физической согласованности.**

Состояние внешней памяти базы данных называется **физически согласованным**, если наборы страниц всех объектов согласованы, т.е. соответствуют состоянию любого объекта либо после его изменения, либо до изменения.

Будем считать, что в журнале отмечаются точки физической согласованности базы данных (point of physical consistency, ppc) – моменты времени, в которые во внешней памяти:

• содержатся согласованные результаты операций, завершившихся до соответствующего момента времени,

• отсутствуют результаты операций, которые не завершились к этому моменту,

• буфер журнала вытолкнут во внешнюю память.Предположим, что удалось восстановить внешнюю память базы данных к состоянию на момент времени tppc ( Теневой механизм , Журнализация постраничных изменений) . Тогда восстановление последнего по времени логически целостного состояния базы данных производится следующим образом (схема восстановления от точки физической согласованности):

**Для транзакции T1** никаких действий производить не требуется. Она закончилась до момента tppc, и все ее результаты гарантированно отражены во внешней памяти базы данных. **Для транзакции T2** нужно повторно выполнить последовательность операций, которые выполнялись после tppc (прямое по смыслу и хронологии выполнение операций транзакции T2). Поскольку транзакция T2 успешно завершилась до момента мягкого сбоя tsf , в журнале содержатся записи обо всех изменениях базы данных, произведенных этой транзакцией.

**Поскольку транзакция T3** не завершилась к моменту мягкого сбоя tsf , при восстановлении необходимо устранить все последствия ее выполнения. Для транзакции T3 нужно выполнить в обратном направлении ту часть операций, которую она успела выполнить до момента tppc.

**Для транзакции T4** , которая успела начаться после момента tppc и закончиться до момента мягкого сбоя tsf , нужно произвести полное повторное выполнение операций в прямом направлении. Поскольку транзакция T4 успешно завершилась до момента мягкого сбоя tsf , в журнале содержатся записи обо всех изменениях базы данных, произведенных этой транзакцией. Наконец, для транзакции T5 , начавшейся после момента tppc и не успевшей завершиться к моменту мягкого сбоя tsf , никаких действий предпринимать не требуется. Результаты операций этой транзакции полностью отсутствуют во внешней памяти базы данных.

72. Способы восстановления физически согласованного состояния (теневой механизм и журнализация постраничных изменений).

**Теневой механизм**

Основная идея состоит в том, что при открытии файла, когда работающая с

файлом программа изменяет блок этого файла, то во внешней памяти выделяется

новый блок, в который записывается новое изменённое содержимое. При этом только

текущая таблица отображения изменяет ссылку на новый блок, а теневая таблица

во внешней памяти остаётся неизменной. При закрытии файла текущая таблица

записывается на место теневой таблицы, и освобождаются ранее изменённые блоки.

Если во время работы с открытым файлом происходит мягкий сбой при котором

теряется содержимое оперативной памяти, то для восстановления согласованного

состояния файла, достаточно просто использовать теневую таблицу отображения.

Как можно применять теневой механизм в контексте баз данных? Вся база

данных хранится в нескольких файлах: в System R они назывались сегментами, в

Oracle они называются table space, но главное, что это аналоги блочных файлов.

Периодически СУБД устанавливает точки физической согласованности базы данных

для каждого файла, при этом:

1) Все логические операции завершаются, а новые не начинаются.

2) Выталкиваются все страницы буферного пула базы данных, содержимое кото-

рых отличается от содержимого соответствующих блоков внешней памяти.

3) Теневая таблица отображения файлов базы данных заменяется текущей таб-

лицей отображения (правильнее сказать, текущая таблица отображения запи-

сывается на место теневой).

Здесь имеется некоторая проблема, состоящая в том, что в любой момент вре-

мени теневая таблица отображения должна быть корректной, т.е. соответствовать

некоторому ранее зафиксированному физически целостному состоянию базы данных.

Для этого необходимо обеспечить атомарность операции замены теневой таблицы

отображения. Т.е. надо, чтобы либо она полностью заменилась, либо отсутствовала

во внешней памяти.

В общем случае таблица отображения может занимать несколько блоков внешней памяти, и для записи текущей таблицы отображения на место теневой таблицы в этом случае потребуется несколько обменов с дисками. Если в промежутке между этими обменами возникнет мягкий сбой, то будет утрачена теневая таблица отображения (поскольку частично её поменяем, а частично оставим её такой, какой она была) и безнадёжно потеряем текущую, т.к. она была в оперативной памяти, т.е. мы просто лишимся возможности восстановить за счет использования последнего физически согласованного состояния базы данных.

Чтобы это не произошло, во внешней памяти поддерживаются две области

хранения таблицы отображения файлов. Будем называть их областями А и В.

Кроме того, в отдельном блоке внешней памяти хранится флаг F, показывающий,

какая из этих областей в данный момент содержит действующую теневую таблицу

отображения. Назовём соответствующие значения флага FA и Fв.

Тогда, если сохраненным во внешней памяти значением флага является FA, то

текущая таблица отображения записывается в область В. После успешной записи

текущей таблицы в область В флаг принимает значение Fв. Считается, что операция

записи одного блока на диск является атомарной:

* Если эта операция заканчивается успешно, это означает, что новая теневая таблица отображения хранится в области В.
* Если же запись текущей таблицы отображения в область В не удалась, или если не выполнилась операция записи блока с флагом F, то продолжает действовать старая теневая таблица отображения.

При использовании теневого механизма восстановление хронологически последнего сохраненного физически согласованного состояния базы данных происходит мгновенно: СУБД устанавливает текущую таблицу отображения равной теневой. Все проблемы восстановления решаются, но за счет слишком большого перерасхода внешней памяти. В пределе может потребоваться вдвое больше внешней памяти, чем реально нужно для хранения базы данных.

Заметим, что чем больше времени между двумя контрольными точками, тем

больше требуется внешней памяти. С другой стороны, нельзя выполнять операцию

установки контрольный точки слишком часто, т.к. она вычислительно дорогая.

Поэтому как правильно выбрать интервалы времени между контрольными точками

- вопрос сложный.

**Журнализация постраничных изменений**

Каждый раз когда в какой-то транзакции при выполнении какой-то операции нужно поменять страницу кэша (буферного пула), которая соответствует копии какого-то блока внешней памяти, формируется запись о постраничном изменении.

Первый этап восстановления после мягкого сбоя состоит в постраничном откате

недовыполненных логических операций. Подобно тому, как это делается с логиче-

скими записями по отношению к транзакциям, последней записью о постраничных

изменениях от одной логической операции является запись о конце операции. Вообще,

выполнение логических операций уровня RSS носит транзакционный характер.

В частности, как уже отмечалось раньше, при выполнении логической операции

обновления базы данных, вообще говоря, изменяется несколько блоков базы данных.

Для обеспечения возможности отката отдельной операции, нужно до конца операции

монопольно блокировать все страницы буферного пула базы данных, содержащие

копии изменяемых этой операцией блоков базы данных. Это может потребоваться,

например, если обнаруживается нарушение свойства уникальности какого-либо

индекса.

Допустим, что база данных в момент мягкого сбоя находилась в физически

согласованно состоянии, но часть изменённых блоков осталась в кэше и во внешнюю

память не попала, а другая часть попала. При восстановлении к точке физически

согласованного состояния нужно убрать следы незавершённых изменений, а для

этого надо уметь различать: для каждой записи в журнале блок базы данных

находится в старом состоянии или в новом, которое соответствует этой записи.

Для того, чтобы с этим разобраться, в каждый блок внешней памяти и в каждую

запись об изменении этого блока в журнале постраничных изменений записывается

монотонно возрастающий номер записи. Чтобы понять, нужно ли применить дан-

ную запись о постраничном изменении соответствующего блока внешней памяти

для восстановления состояния этого блока, требуется всего лишь сравнить номер,

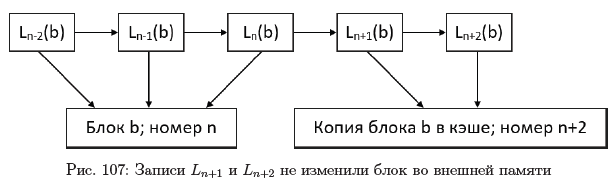
содержащийся в этом блоке, с номером, содержащимся в журнальной записи. Если в

блоке содержится номер, меньший номера журнальной записи, то это означает, что

буферная страница, в которой выполнялось соответствующее изменение, не была к

моменту мягкого сбоя вытолкнута во внешнюю память, и применять данную запись

для восстановления соответствующего блока внешней памяти не требуется.



На рис. 107 изображена цепочка записей в журнале об изменении блока b. В

самом блоке b указан номер п, который означает, что в состоянии блока отражены

результаты операций изменения блока, соответствующих журнальным записям с

номерами п - 2, п - 1, п. Другими словами, записи с номерами п - 2, п - 1, п попали

во внешнюю память, а с номерами п + 1, n + 2 не попали.

Изменения блока, произведённые операциями, которым соответствуют две хро-

нологически последние журнальные записи Ln+1(b) и Ln+2(b), в его состоянии во

внешней памяти не отражены, поскольку не было выполнено выталкивание во внеш-

нюю память страницы буферного пула, содержащей копию блока b. Поэтому при

восстановлении состояния блока требуется выполнить обратные операции изменения

блока b, соответствующие журнальным записям Ln(b), Ln-1(b) и Ln-2(b).

## 74. Объектно-ориентированная модель данных. Ее структурная, манипуляционная и целостная части. Реализации

Объектная БД – набор контейнеров произвольного типа (объекты)

Типы данных:

Литеральные:

1. атомарные

* числовые
* строковые
* темпоральные
* логические

1. конструированные

* структуры
* коллекции (мн-ва, мульти мн-ва, массивы, списки, словари)

Объектные:

1. атомарные
2. объект: коллекция

Литеральные – только в качестве типов атрибутов внутри объекта значение д. типов не имеют идентификаторов и не могут самостоятельно храниться в БД.

Объектные – у любого типасуществует конструктор (созд. и инициал.) и отдельный атрибут OID (объект. идентификатор). OID уникален по всей БД (ключ – только в пределах таблицы). OID генерирует БД и не зависит от состояния объекта (ключ зависит от состояния объекта)

3 вида эквивалентностей объектов:

* идентичность (идентификаторы совпадают)
* эквивалентность по значению (значение всех их атрибутов одинаково, в случае сложных объектов - рекурсивно)
* поверхностная эквивалентность, когда атрибуты идентичны (простые – по значению, объектные – по идентификатору)

Объектные коллекции могут храниться в БД самостоятельно и имеют идентификатор.

Манипулирование данными

Операторы могут вызываться из любого языка программирования и наоборот

В OQL результатом запроса является:

* индивидуальный объект
* коллекция объектов
* коллекция литеральных значений
* индивидуальные литеральные значения

ODL позволяет описать схему БД в виде набора интерфейсов объектных типов.

Подходы к созданию объектов:

1. устойчивость – характеристика всех экземпляров всех объектных типов
2. вызов конструктора предпол. создание транзитного экземпляра
3. смешанный: тип объекта определяется параметрами конструктора

Удаление объектов:

1. объект физически удаляется в любом случае
2. объект имеет счетчик ссылок. При ненулевом значении счетчика выставляется флаг уничтожения, а объект физически удаляется после того, как счетчик=0.

Ограничения целостности:

объекты совпадают тогда и только тогда, когда они идентичны, значит, нет ограничения целостности как в реляц. БД. Не существует потомков без предка.

Реализации: Versant, Objectivity DB, db4objects (Java/.NET)

с середины 2000x ODBMS.ORG – 2008 – новый стандарт

# **75. Объектно-реляционные расширения языка sql. Возможные подходы к объектно-реляционному отображению без использования объектно-реляционных расширений sql.**

Определение пользователем типов данных и типизированных таблиц.

UDT:

* Индивидуальный тип. Основан на единственном предопределенном типе (типа typedef), но без наследственных операций, их нужно указывать.
* Структурный тип. Именованный тип данных, включающий 1 или более атрибутов любых из допустимых в SQL типов данных, в том числе в данных структурного типа можно использовать механизм наследственности от ранее определенного структурного типа.

При определении типизированных таблиц указывается ранее определенный структурный тип, и если в нем N атрибутов, то в таблице N+1 столбец “лишний” – самоссылающийся и содержит типизированный уникальный идентификатор строк.

Способ генерации значений при определении структурного типа

1. SYSTEM GENERATED – аналог ООБД
2. USER DEFINED (TYPE) – опр. тип уникального идентификатора, пользователь указывает значение, когда вставляет строки
3. USER DEFINED (список столбцов) – аналог ключа

## 76 Истинная реляционная модель данных. Ее структурная, манипуляционная и целостная части. Реализации.

Типы данных

* скалярный (инкапсулир. тип, внутренняя структура которого скрыта от пользователя)
* кортежный (определяется указанием заголовка в вие множества пар <имя атр., тип атр.>, значение – триплет <имя, тип, значение>)
* отношение (определяется заголовком <имя, тип>, значение-заголовок, совпадение заголовка в определении + тело как множество кортежей)

Любой кортеж в отношении содержит ровно 1 значение для любого атрибута, значит, отношение в 1NF.

БД – набор долговременно хранимых отношений.

Манипуляция данными:

* алгебра А
* язык D (принципы: для запросов – алгебраический подход, запросы, адрес к сложным данным, формулируется более точно, чем на SQL.)

Ограничения целостности:

1. обязательное определение хотя бы 1 возможного ключа
2. поддержка ограничения целостности как произвольных условных выражений, которые логически эквивалентны замкнутой правильно построенной формуле.
3. Рекомендуется использовать ср. под. ссылоч. целостности DATAPHOR.