Базы данных (20.01.23)

Вопросы к экзамену

[Основы баз данных. Карпова И.П. 2009 МФТИ]

Планы запросов - это просто! Разбор оптимизаций запросов PostgreSQL на живых при…

Базы данных, знаний и хранилище данных. Бигдата, субд sql и nosql

1. Модели данных баз данных с примерами промышленных СУБД.

Глава 2 в карповой. очень подробно

- **Иерархическая – дерево (*Cache*)** - Реализация связей типа n:m не поддерживается. каждая запись идентифицируется полным сцепленным ключом, который образуется путём конкатенации всех ключей экземпляров родительских записей. Основным недостатком ИМД является дублирование данных. Оно вызвано тем, что каждая сущность (атрибут) может относиться только к одной родительской сущности.
- **Сетевая – граф (neo4j)** - В этой модели не обеспечивается физическая независимость данных, т.к. наборы организованы с помощью физических ссылок. Также в СМД не обеспечивается независимость данных от программ. Из-за этих недостатков эта модель не получила широкого распространения.
- **Реляционная – (Postgres, MySQL, …)** базис с использованием теории множеств. Таблицы и отношения между ними.
- **ОРМД** - надстройка над РМД. Добавили объекты и парадигму ооп (наследование, инкапсуляция тырыпыры)
- **Объектно-ориентированная - (MongoDB, Cassandra, …)**

2. Нормализация (нормальные формы) в реляционной модели данных. МОГУ УМЕЮ ПРАКТИКУЮ

1 форма: все ключи простые

2 форма: все неключевые поля полностью зависят от ключевого поля целиком.

3 форма: отсутствие транзитивной зависимости

3 форма Бойса-Кодда: ключевые не должны зависеть от неключевых

4 форма: не должно быть нетривиальных многозначных зависимостей

3. Первичный ключ, внешний ключ, отношения.

4. Язык SQL (SELECT, WHERE, GROUP BY, HAVING).

[where vs having]

Where - для фильтрации строк

Having - для фильтрации групп

5. Язык SQL (DML: INSERT, UPDATE, DELETE). Варианты синтаксиса для множественного обновления данных.

```
INSERT INTO <table.name>(Column1, Column2, Column3)
VALUES (Value11, Value12, Value13),
       (Value21, Value22, Value23);


UPDATE <table.name>
SET Column1 = Value1, Column2 = Value2
WHERE <condition>


DELETE FROM <table.name>
WHERE <condition>
```

6. Язык SQL (Триггеры, процедуры, функции, а также курсоры, циклы, условные операторы, временные таблицы).

Plpgsql

Создание Триггера:

```
CREATE FUNCTION emp_stamp() RETURNS trigger AS $emp_stamp$
    BEGIN
        NEW – строка, которую вставили или изменили
        RETURN NEW;
    END;
$emp_stamp$ LANGUAGE plpgsql;
CREATE TRIGGER emp_stamp BEFORE INSERT OR UPDATE ON emp
    FOR EACH ROW EXECUTE PROCEDURE emp_stamp();
```

Создание SQL процедуры:

```
CREATE PROCEDURE insert_data(a integer, b integer)

LANGUAGE SQL

AS $$

INSERT INTO tbl VALUES (a);

INSERT INTO tbl VALUES (b);

$$;

CALL insert_data(1, 2);
```

Создание pgplsqpl процедуры

Создание pgplsqpl функции

RETURN NEXT - добавляет строку в результирующую таблицу

Можно также возвращать результаты запроса с помощью RETURN QUERY

Курсоры:

Вместо того чтобы сразу выполнять весь запрос, есть возможность настроить курсор, инкапсулирующий запрос, и затем получать результат запроса по нескольку строк за раз. Одна из причин так делать заключается в том, чтобы избежать переполнения памяти, когда результат содержит большое количество строк. (Пользователям PL/pgSQL не нужно об этом беспокоиться, так как циклы FOR автоматически используют курсоры, чтобы избежать проблем с памятью.) Более интересным вариантом использования является возврат из функции ссылки на курсор, что позволяет вызывающему получать строки запроса. Это эффективный способ получать большие наборы строк из функций.

```
DECLARE

    curs1 refcursor;

    curs2 CURSOR FOR SELECT * FROM tenk1;

    curs3 CURSOR (key integer) FOR SELECT * FROM tenk1 WHERE unique1 = key;
```

FETCH - получать данные из курсора

```
FETCH curs1 INTO rowvar;
```

MOVE - двигать курсор

Временная таблица:

CREATE TEMPORARY TABLE

С таким указанием таблица создаётся как временная. Временные таблицы автоматически удаляются в конце сеанса или могут удаляться в конце текущей транзакции (см. описание ON COMMIT ниже). Существующая постоянная таблица с тем же именем не будет видна в текущем сеансе, пока существует временная, однако к ней можно обратиться, дополнив имя указанием схемы. Все индексы, создаваемые для временной таблицы, также автоматически становятся временными.

7. Индексы в Реляционных СУБД, виды индексов.

ВСЕ ПРО ИНДЕКСЫ

Плюсы: быстрый поиск, сортировка.

Минусы: время

Способы представления индекса:

- B-Tree
- Пространственные индексы
    - Для работы с пространственными данными
    - R-tree
    - Spatial grid index
- Хеш-индексы
- Bitmap индекс
- Реверсивный индекс
    - то же самое, что и btree, только ключ реверсируется. чтобы было распределение данных по дереву
- Инвертированный индекс
- Частичный индекс

Виды индексов:

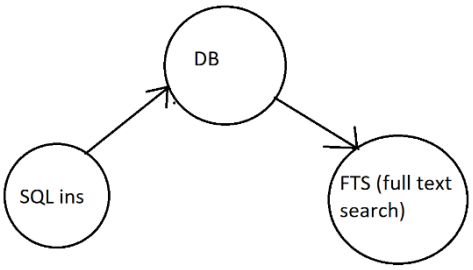
- Уникальные / неуникальные
- Составные (важен порядок следования)
- Кластерные

    - Существует два типа индексов: кластерные и некластерные. При наличии кластерного индекса строки таблицы упорядочены по значению ключа этого индекса. Если в таблице нет кластерного индекса, таблица называется кучей[3]. Некластерный индекс, созданный для такой таблицы, содержит только указатели на записи таблицы. Кластерный индекс может быть только одним для каждой таблицы, но каждая таблица может иметь несколько различных некластерных индексов, каждый из которых определяет свой собственный порядок следования записей.

Пример антипаттерна с индексами:

Проблема: (like `% some-text %`)

Решение: Full text search



8. Сбалансированное дерево, как пример индекса РСУБД.

B-Tree – дерево порядка n (из каждой вершины n ребер) Привести пример построения

Свойства B-Tree:

- Все конечные узлы расположены на одном уровне.
- Каждый узел может содержать n потомков и n-1 значений. Ссылка влево – на значение меньшее, ссылка вправо – на значение большее.
- Любая не конечная вершина имеет *окр.вверх(n / 2)* потомков.
- Если вершина содержит k значений, то у нее k+1 потомков.

Вопрос: Почему нельзя построить индекс для поля varchar max?

Ответ: Нельзя рассчитать глубину дерева

9. Оптимизация запросов. Планы запросов.

Последовательность выполнения запросов в реляционных СУБД:

1. **Лексический и синтаксический анализ. В результате вырабатывается внутреннее представление запроса.**
2. **Логическая оптимизация**
3. **Построение планов запроса и выбор оптимального (близкого к оптимальному). Оптимизатор использует информацию о существующих путях доступа к данным.**
4. **Формируется процедурное представление выбранного плана.**
5. **Непосредственное выполнение запроса.**

Ранговая оптимизация: При использовании этого метода план составляется на основании существующих путей доступа и их рангов. Все пути доступа ранжируются на основании

знаний о правилах и последовательности осуществления этих путей. Ранг пути доступа определяется на основании знаний о последовательности реализации этого пути. Метод оптимизации по синтаксису учитывает ранги путей доступа. Если для какой-либо таблицы существует более одного пути доступа, то выбирается тот путь, чей ранг выше, т.к. при прочих равных условиях он выполняется быстрее, чем путь с более низким рангом

Метод оптимизации, основанный на стоимости: При расчёте стоимости оптимизатор может учитывать такие параметры, как количество необходимых ресурсов памяти, время операций дискового ввода-вывода, время процессора. Стоимость плана выполнения запроса определяется на основании сведений о распределении данных в таблицах, к которым обращается команда, и связанных с ними кластеров и индексов

Посмотреть план запроса: EXPLAIN SELECT * …….

На что обращать внимание:

- Время выполнение запроса
- Кол-во операций ввода / вывода
- Объем обрабатываемых данных (стоимость операций)

Операции (*цвета – группировка операций*):

- Table scan
- Index scan (просмотр всего индекса)
- Index seek (поиск в индексе)
- Nested loops (вложенные циклы)
- Hash join (хеш-соединение)
- Merge join (соединение слиянием)
- Sort
- Агрегаты, группировки, параллелизм

Что уметь:

- Делать в какой-либо программе для СУБД (PGAdmin)
- Как получить план запросов
- Как изменить план давлением индекса
- Изменить план запросов с помощью изменения запроса

10. Транзакции. ACID.

Транзакция – это упорядоченная последовательность операторов обработки данных, которая переводит базу данных из одного согласованного состояния в другое.

Транзакция обладает следующими свойствами:

1. Логическая неделимость (атомарность, Atomicity) означает, что выполняются либо все операции (команды), входящие в транзакцию, либо ни одной. Система гарантирует невозможность запоминания части изменений, произведённых транзакцией. До тех пор,

пока транзакция не завершена, её можно "откатить", т.е. отменить все сделанные командами транзакции изменения. Успешное выполнение транзакции (фиксация) означает, что все команды транзакции проанализированы, интерпретированы как правильные и безошибочно исполнены.

2. Согласованность (Consistency): транзакция начинается на согласованном множестве данных и после её завершения множество данных согласовано. Состояние БД является согласованным, если данные удовлетворяют всем установленным ограничениям целостности и относятся к одному моменту в состоянии предметной области.

3. Изолированность (Isolation), т.е. отсутствие влияния транзакций друг на друга. (На самом деле это влияние существует и регламентируется стандартом: см. раздел 5.3. "Уровни изоляции транзакций").

4. Устойчивость (Durability): результаты завершённой транзакции не могут быть потеряны. Возврат БД в предыдущее состояние может быть достигнут только путём запуска компенсирующей транзакции. Транзакции, удовлетворяющие этим свойствам, называют ACID-транзакциями

Система записывает изменения в БД, не дожидаясь завершения транзакции, а старые значения данных сохраняет на время выполнения транзакции в сегментах отката.

Для сохранения сведений о транзакциях СУБД ведёт **журнал транзакций**. **Журнал транзакций** – это часть БД, в которую поступают данные обо всех изменениях всех объектов БД. Журнал недоступен пользователям СУБД и поддерживается особо тщательно (иногда ведутся две копии журнала, хранимые на разных физических носителях).

Форма записи в журнал изменений зависит от СУБД. Но обычно там фиксируется следующее:

- номер транзакции (номера присваиваются автоматически по возрастанию);
- состояние транзакции (завершена фиксацией или откатом, не завершена, находится в состоянии ожидания);
- точки сохранения (явные и неявные);
- команды, составляющие транзакцию, и проч.

Внесение изменений в журнал транзакций всегда носит **опережающий характер** по отношению к записи изменений в основную часть БД (протокол **WAL – Write Ahead Log**). Эта стратегия заключается в том, что запись об изменении любого объекта БД должна попасть во внешнюю память журнала транзакций раньше, чем изменённый объект попадёт во внешнюю память основной части БД. Если СУБД корректно соблюдает протокол WAL, то с помощью журнала транзакций можно решить все проблемы восстановления БД после сбоя, если сбой препятствуют дальнейшему функционированию системы (например, после сбоя приложения или фонового процесса СУБД).

11. Уровни изоляции транзакций. Модели конкурентного доступа.

В общем случае взаимовлияние транзакций может проявляться в виде:

- потери изменений;
- чернового чтения;
- неповторяемого чтения;
- фантомов

СУБД не допускает такого взаимовлияния транзакций, при котором возможна потеря изменений!

Уровни изоляции:

- Read Uncommitted – чтение незавершенных транзакций
- Read Committed – чтение завершенных транзакций
- Repeatable Read – повторяемое чтение
- Serializable – последовательное чтение

По умолчанию в СУБД обычно установлен уровень Read Committed.

Модели конкурентного доступа:

- Пессимистические алгоритмы: Блокировки. Блокировка – это временное ограничение доступа к данным, участвующим в транзакции, со стороны других транзакций.. Блокировка относится к пессимистическим алгоритмам, т.к. предполагается, что существует высокая вероятность одновременного обращения нескольких пишущих транзакций к одним и тем же данным.
- Оптимистические: временные отметки. Использование временных отметок относится к оптимистическим алгоритмам разграничения транзакций. Для их эффективного функционирования необходимо, чтобы вероятность одновременного обращения нескольких пишущих транзакций к одним и тем же данным была невелика. Временная отметка – это уникальный идентификатор, который СУБД создаёт для обозначения относительного момента запуска транзакции.

Во всех случаях обнаружения конфликта система перезапускает текущую транзакцию с более поздней временной отметкой. Если конфликта нет, то транзакция выполняется

12. Блокировки и взаимоблокировки.

Типы блокировок:

- по степени доступности данных: разделяемые и исключающие;
- по множеству блокируемых данных: строчные, страничные, табличные;
- по способу установки: автоматические и явные.

Разделяемая блокировка, установленная на определённый ресурс, предоставляет транзакциям право коллективного доступа к этому ресурсу. Обычно этот вид блокировок используется для того, чтобы запретить другим транзакциям производить необратимые изменения. Например, если на таблицу целиком наложена разделяемая блокировка, то ни одна транзакция не сможет удалить эту таблицу или изменить её структуру до тех пор, пока

эта блокировка не будет снята. (При выполнении запросов на чтение обычно накладывается разделяемая блокировка на таблицу.)

Исключающая блокировка предоставляет право на монопольный доступ к ресурсу. Исключающая (монопольная) блокировка таблицы накладывается, например, в случае выполнения операции ALTER TABLE, т.е. изменения структуры таблицы. На отдельные записи (блоки) монопольная блокировка накладывается тогда, когда эти записи (блоки) подвергаются модификации.

Явную блокировку также можно наложить с помощью ключевых слов *for update*

И явные, и неявные блокировки снимаются при завершении транзакции.

Тупиковые ситуации (deadlocks) возникают при взаимных блокировках транзакций, которые выполняются на пересекающихся множествах данных

Стратегии разрешения проблемы взаимной блокировки:

- Транзакция запрашивает сразу все требуемые блокировки
- СУБД отслеживает возникающие тупики и отменяет одну из транзакций с последующим рестартом через случайный промежуток времени.
- Вводится таймаут (time-out) – максимальное время, в течение которого транзакция может находиться в состоянии ожидания. Если транзакция находится в состоянии ожидания дольше таймаута, считается, что она находится в состоянии тупика, и СУБД инициирует её откат с последующим рестартом через случайный промежуток времени.

13. Безопасность БД.

Защита данных включает предупреждение случайного или несанкционированного доступа к данным, их изменения или разрушения со стороны пользователей или при сбоях аппаратуры. Реализация защиты включает:

- контроль достоверности данных с помощью ограничений целостности;
- обеспечение безопасности данных (физической целостности данных);
- обеспечение секретности данных.

Виды сбоев:

- Сбой предложения
- Сбой пользовательского процесса.
- Сбой процесса сервера.
- Сбой носителя (диска).

Резервное копирование означает периодическое сохранение файлов БД на внешнем запоминающем устройстве. Оно выполняется тогда, когда состояние файлов БД является непротиворечивым. Резервная копия (РК) не должна создаваться на том же диске, на

котором находится сама БД, т.к. при аварии диска базу невозможно будет восстановить. В случае сбоя (или аварии диска) БД восстанавливается на основе последней копии.

Все изменения, произведённые в данных после последнего резервного копирования, утрачиваются; но при наличии **архива журнала транзакций** их можно выполнить ещё раз, обеспечив полное восстановление БД на момент возникновения сбоя. Дело в том, что журнал транзакций содержит сведения только о текущих транзакциях. После завершения транзакции информация о ней может быть перезаписана. Для того чтобы в случае сбоя обеспечить возможность полного восстановления БД, необходимо вести архив журнала транзакций, т.е. сохранять копии файлов журнала транзакций вместе с резервной копией базы данных.

Потенциальные ситуации в результате сбоя:

- Транзакция подтверждена, но блоки измененных данных еще не записаны в файл.
- Блоки данных содержат неподтвержденные изменения (так как не дожидаясь подтверждения транзакции, СУБД переписывает на диск модифицированные блоки (при формировании контрольной точки))

Решение:

- Прокрутка вперед
- Прокрутка назад

СУБД хранит информацию о пользователях, логин/пароль. Если данные особо секретны, то СУБД может зашифровать БД, чтобы к ней не подключились через другую СУБД.

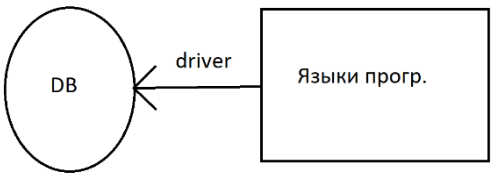
Есть роли пользователей

14. Архитектура ИС (клиент-сервер, трехзвенная) с точки зрения подключения к СУБД.

http://it-claim.ru/Education/Course/ISDevelopment/Lecture_3.pdf

Клиенты бывают – Толстые и Тонкие???

Схема взаимодействия БД и кода:



CAP theorem

Микросервисы / монолит

Безопасность:

- Контроль доступа
    - Встроенные роли
    - (permissions) Разрешения на объекты (таблицы, хранимые процедуры / функции, представления (view))
- Аудит
- Резервное копирование ACID
- Безопасность приложения (SQL атаки)

15. Обзор какой-либо NOSQL СУБД (MongoDB, Cassandra, Cache или любую другую по желанию).

***(для чего нужна, чем отличается от рел.субд + и -, с точки зрения ACID)**

habr.com/ru/post/155115

habr.com/ru/post/152477

Практические задачи

Выбрать пересекающиеся интервалы:

Таблица: Parts (id, start_pos, end_pos)

Решение:

```
SELECT *
FROM parts p1
JOIN parts p2
ON TRUE
WHERE (p1.end_pos >= p2.start_pos)
OR (p2.end_pos >= p1.start_pos);
```

Вывести натуральные числа от 0 до 100 000:

Решение:

```
SELECT *
FROM generate_series(0, 100000);
```

Удалить повторяющиеся значения*:**

Таблица: Persons (id, title, date_of_birth, sex)

Решение:

```
DELETE
FROM Persons
WHERE id NOT IN
(SELECT MIN(Id) FROM Persons
GROUP BY (title, date_of_birth, sex))
```

Размножить записи указанное число раз:

Таблица: Books (id, title, quantity)

Решение: TODO

```
Что изменится если перенести условие из WHERE в JOIN ON??

SELECT T1.*

FROM T1

LEFT JOIN T2

ON T1.id = T2.id

WHERE T1.fed = "AAA" AND T2.fed = "BBB";

# Количество строк будет другое в результате? Надо смотреть
порядок выполнения операция where идет после джоина скорее
всего.
```