1. **Что такое индекс, для чего он нужен?**

Индексы - распространенный способ повышения производительности базы данных. Индекс позволяет серверу базы данных искать и получать конкретные строк гораздо быстрее, чем без них. При отсутствии предварительной подготовки, система будет сканировать всю таблицу, строку за строкой, чтобы найти все соответствующие записи. Если в таблице много строк и только несколько строк (может быть ноль или одна), которые будут возвращены на такой запрос, то это явно неэффективный метод. Но, если системе было поручено поддерживать индекс столбца, то она может использовать более эффективный метод для обнаружения подходящих строк.

Наиболее значительные проблемы с производительностью вызываются отсутствием нужных индексов. Поэтому столкнувшись с медленным запросом, в первую очередь проверьте, существуют ли индексы, которые он может использовать. Если нет -- постройте их.

Поля, являющиеся внешими ключами, и поля, по которым объединяются таблицы, индексировать надо обязательно.

1. **Какие бывают типы индексов, в чем их различия?**

PostgreSQL предлагает несколько типов индексов: B-tree, HASH, GiST и GIN. Каждый тип индекса используется свой алгоритм, который лучше всего подходит для различных типов запросов. По умолчанию, команда CREATE INDEX создает B-tree индексы, которые соответствуют наиболее распространенным ситуациям.

B-tree может обрабатывать запросы равенства и интервала данных, которые должны быть отсортированы в каком-нибудь порядке. В частности, планировщик запросов PostgreSQL будет рассматривать вопрос об использовании индекса B-tree, когда индексированный столбец участвует в сравнении с помощью одного из этих операторов: < <= = >= >

Hash-индексы могут обрабатывать только простые сравнения равенства. Планировщик запросов рассматривает вопрос об использовании данного индекса, когда индексированный столбец участвует в сравнении с помощью оператора "="'. (Hash-индексы не поддерживают поиска по IS NULL.) Следующая команда используется для создания hash-индекса:

**CREATE** **INDEX** name **ON** **TABLE** UNING hash (**COLUMN**);

GiST-индекс - представляет собой некую инфраструктуру, в которой могут быть реализованы много различных стратегий индексирования . Таким образом, конкретные операторы, с которыми могут быть использованы GiST-индексы, варьируются в зависимости от стратегии индексирования (класса оператора).

GIN-индексы - "перевёрнутые" индексы, которые могут обрабатывать значения, содержащие более одного ключа, например, массивов. Как и GiST, GIN поддерживает множество различных пользовательских стратегий индексирования и конкретных операторов, с которыми GIN-индекс может быть использован в зависимости от стратегии индексирования.

1. **Для чего нужны функциональные индексы?**

Вы можете построить индекс не только по полю/нескольким полям таблицы, но и по выражению, зависящему от полей. Пусть, например, в вашей таблице foo есть поле foo\_name, и выборки часто делаются по условию «первая буква foo\_name = 'буква', в любом регистре». Вы можете создать индекс

CREATE INDEX foo\_name\_first\_idx

ON foo ((lower(substr(foo\_name, 1, 1))));

и запрос вида

SELECT \* FROM foo

WHERE lower(substr(foo\_name, 1, 1)) = 'ы';

будет его использовать.

1. **Какой формат имеет команда EXPLAIN?**

http://postgrespro.ru/doc/using-explain.html

EXPLAIN SELECT и дальше вся структура нужного селекта

1. **Для чего нужна команда EXPLAIN?**

Команда EXPLAIN [запрос] показывает, каким образом PostgreSQL собирается выполнять ваш запрос. Команда EXPLAIN ANALYZE [запрос] выполняет запрос и показывает как изначальный план, так и реальный процесс его выполнения.

1. **В чем заключаются различия PRIMARY KEY и UNIQUE?**

Ограничения уникальности дают уверенность в том, что данные, содержащиеся в колонке или группе колонок являются уникальными по отношению к другим строкам в той же таблице. Синтаксис:

CREATE TABLE products (

product\_no integer **UNIQUE**,

name text,

price numeric

);

для случая, когда это ограничение записывается на колонку и

CREATE TABLE products (

product\_no integer,

name text,

price numeric,

**UNIQUE (product\_no)**

);

когда это ограничение записывается на таблицу.

Если ограничение уникальности ссылается на группу колонок, эти колонки перечисляются через запятую:

CREATE TABLE example (

a integer,

b integer,

c integer,

**UNIQUE (a, c)**

);

Такая запись говорит, что комбинация значений в указанных колонках является уникальной для всей таблицы, хотя любое одно из значений этих колонок может не быть (и обычно не является) уникальным.

Своё собственное имя для ограничения уникальности вы можете назначить обычным способом:

CREATE TABLE products (

product\_no integer **CONSTRAINT must\_be\_different** UNIQUE,

name text,

price numeric

);

Добавление ограничения уникальности автоматически создаст уникальный btree индекс на колонку или группу колонок, использующих данное ограничение.

Ограничение уникальности нарушается, когда в таблице есть более, чем одна строка, с одинаковыми значениями колонок, включённых в это ограничение. Однако, два значения null при сравнении не признаются равными друг другу. Это означает, что даже если значения null есть в самом ограничении уникальности, можно хранить дублирующиеся строки, которые содержат значения null по крайней мере в одной из колонок, которая указывается в ограничении. Такое поведение соответствует стандарту SQL, но мы имеет сведения, что другие SQL СУБД могут не следовать этому правилу. Так что будьте осторожны, когда разрабатываете приложения, которые задумываются как переносимые.

==========================================================

Технически, ограничение первичного ключа просто является комбинацией ограничений уникальности и не-null. Таким образом, следующие определения двух таблиц, позволяют вводить одинаковые данные:

CREATE TABLE products (

product\_no integer UNIQUE NOT NULL,

name text,

price numeric

);

CREATE TABLE products (

product\_no integer **PRIMARY KEY**,

name text,

price numeric

);

Первичные ключи также могут ограничивать более чем одну колонку; синтаксис сходен с ограничением уникальности:

CREATE TABLE example (

a integer,

b integer,

c integer,

**PRIMARY KEY (a, c)**

);

Первичный ключ говорит, что колонка или группа колонок могут быть использованы как уникальный идентификатор строки в таблице. (Это прямое назначение первичного ключа. Заметим, что ограничение уникальности не может выполнять такую функцию само по себе, потому что не исключает значения null). Такой идентификатор полезен как для целей документирования, так и для клиентских приложений. Например, приложение с графическим интерфейсом, которое позволяет изменять значения строк наверняка будет нуждаться в первичном ключе таблицы, чтобы идентифицировать нужные строки.

Добавление первичного ключа автоматически создаст уникальный btree индекс на колонку или группу колонок, используемых как первичный ключ.

Таблица может иметь не более одного первичного ключа. (В ней может быть любое количество ограничений уникальности и ограничений не-null, которые функционально являются тем же, но только одно из этих ограничений может быть индентифицировано как первичный ключ.) Теория реляционных баз данных требует наличия первичного ключа для каждой таблицы. PostgreSQL не заставляет выполнять это правило, но лучше всего следовать ему.

1. **Почему не выгодно создавать редко используемые индексы?**

После того, как индекс будет создан, система должна поддерживать ее синхронизированным с таблицей. Это добавляет дополнительные расходы на операции с данными. Поэтому индексы, которые редко или никогда не используются в запросах должны быть удалены.

1. **Какой формат имеет команда CREATE INDEX?**

**CREATE** **INDEX** имя\_индекса **ON** имя\_таблицы (имя\_поля\_в\_этой\_таблице);

1. **В чем заключается отличие ограничений таблиц от ограничений полей?**

Ограничения полей – это условие, накладываемое на поле (например не больше чего-то там), а ограничение таблицы – это сравнение полей (поле1 *оператор\_сравнения* поле2)

1. **Как можно ограничить поле таким образом, чтобы данные в нем соответствовали заданным условиям?**

Ограничение целостности check является наиболее часто используемым видом ограничения. Оно позволяет задать для определённой колонки, выражение, которое будет осуществлять проверку, помещаемого в эту колонку значения. Если значение удовлетворяет, заданному ограничению, то выражение должно возвращать Логическое значение (истина). Например, если требуется, чтобы в колонке были только положительные цены товаров, вы можете использовать:

CREATE TABLE products (

product\_no integer,

name text,

price numeric **CHECK (price > 0)**

);

Как видите, определение ограничения следует после указания типа данных, как и определение значения по умолчанию. Значения по умолчанию и ограничения могут следовать в произвольном порядке. Ограничение check состоит из ключевого слова CHECK, за которым следует выражение в круглых скобках. Выражение в ограничении check должно использовать ту колонку, на которую оно накладывается, в противном случае это ограничение не будет иметь смысла.

Вы можете дать ограничению конкретное имя. Это сделает более понятными сообщения об ошибках и позволит вам ссылаться на это ограничение, когда вам понадобится его изменить. Синтакис такой:

CREATE TABLE products (

product\_no integer,

name text,

price numeric **CONSTRAINT positive\_price** CHECK (price > 0)

);

Таким образом, для создания именованого ограничения, используйте ключевое слово CONSTRAINT, за которым следует индентификатор (имя), после которого следует, собственно, определение ограничения. (Если при использовании этого синтаксиса вы не укажете имя ограничения, СУБД выберет имя за вас.)

Ограничение check может также накладываться на несколько колонок. Допустим, что вы храните обычную цену и цену со скидкой и хотите иметь уверенность в том, что цена со скидкой меньше, чем обычная цена.

CREATE TABLE products (

product\_no integer,

name text,

price numeric CHECK (price > 0),

discounted\_price numeric CHECK (discounted\_price > 0),

**CHECK (price > discounted\_price)**

);

Первые два ограничения нам уже знакомы. Третье использует новый синтаксис. Оно не привязано к определённой колонке, а появляется как отдельный элемент в списке колонок, разделённых запятой. Определения колонок и такие определения ограничений могут следовать в смешанном порядке.

1. **Что делает опция DISTINCT команды SELECT?**

Секция *DISTINCT* определяет поле или выражение, значения которого должны входить в итоговый набор не более одного раза.

1. **Как включить в результаты выборки все поля таблицы?**

Select \* from tablename;

1. **Как можно отсортировать результаты выборки?**

После законченного WHERE пишем ORDER BY поле1,поле2,…

*ORDER BY выражение*. Сортировка результатов команды *SELECT* по заданному *выражению*.

*[ASC | DESC | USING оператор]*. Порядок сортировки, определяемой секцией *ORDER BY* *выражение*: по возрастанию (ASC) или по убыванию (DESC). С ключевым словом *USING* может задаваться *оператор*, определяющий порядок сортировки (например, < или >).

1. **Для чего нужна секция WHERE команды SELECT?**

Секция *WHERE* ограничивает итоговый набор заданными критериями. Условие должно возвращать простое логическое значение (true или false), но оно может состоять из нескольких внутренних условий, объединенных логическими операторами (например, AND или OR).

1. **Что такое подзапрос, для чего он нужен?**

Внутренний селект, чтобы задать поиск соответствий среди некоторой выборки, см. пример в отчете

1. **Как выглядит структура команды SELECT?**

*SELECT [ALL | DISTINCT [ON (выражение [...])]]*

*цель [AS имя] [...] [FROM источник [...]]*

*[[NATURAL] тип\_объединения источник*

*[ON условие | USING (список\_полей)]] [...]*

*[WHERE условие] [GROUP BY критерий [...]]*

*[HAVING условие [...]]*

*[{UNION | INTERSECT | EXCEPT} [ALL] подзапрос]*

*[ORDER BY выражение [ASC | DESC | USING оператор] [...]]*

*[FOR UPDATE [OF таблица [...]]]*

*[LIMIT {число | ALL} [OFFSET начало]]*

1. **Можно ли в результаты выборки добавлять поля, которых нет в таблице?**

Можно, например если делать производное поле, типа …, (поле1+поле2)/2 AS имя\_поля, …

1. **Каким образом можно изменить имена полей, являющихся результатом выборки?**

Через синонимы, имя\_поля AS новое\_имя

1. **Что такое точечная запись, для чего она нужна?**

Для предотвращения неоднозначности в «полные» имена столбцов включается имя таблицы. При этом используется специальный синтаксис, называемый точечной записью (название связано с тем, что имя таблицы отделяется от имени поля точкой). Типа имя\_таблицы.имя\_столбца. Нужно в тех случаях, если в нескольких таблицах есть столбцы с одинаковыми названиями. Точечная запись обязательна только при наличии неоднозначности между наборами данных.