Лабораторная работа 3

Цели лабораторной работы:

* научиться анализировать план выполнения запросов и уменьшать сложность выполнения запросов,
* научиться анализировать состояние сервера СУБД, обучиться основам администрирования.

**Содержание**

* 1 Введение
* 2 Создание индексов
  + 2.1 Типы индексов
  + 2.2 Функции в индексах
  + 2.3 Индексы на несколько колонок
  + 2.4 Уникальный индекс
  + 2.5 Условные индексы
* 3 Внешние ключи
* 4 План запроса. EXPLAIN
  + 4.1 Основы EXPLAIN
  + 4.2 EXPLAIN ANALIZE
  + 4.3 EXPLAIN (ANALYZE, BUFFERS)
* 5 Дополнительно
* 6 Параметры и состояние PostgreSQL-сервера

Введение

Ваша задача в этой лабораторной работе:

* С помощью EXPLAIN посчитать сложность выполнения комплексных запросов из прошлой лабораторной работы,
* Разобраться с выводом EXPLAIN и суметь объяснять, какие части подзапроса сколько будут выполняться, сколько данных будет задействовано.
* С помощью индексов ускорить выполнение запросов.
* Проанализировать выполнение запроса после создания индексов и сравнить планы выполнений.

Создание индексов

Индексы нужны для того, чтобы ускорить доступ к данным. Для каждой таблицы доступен только один самый быстрый индекс - первичный ключ. Как правило, это id в каждой таблице. Запросы, в условиях которых фильтрация происходит только по id, работают быстро, таблица при этом не сканируется целиком.

В таблицу также можно добавить еще индексы для ускорения выполнения запросов. Они могут относиться как к одному, так и к группе атрибутов. Индексы выбирают исходя из информационной потребности пользователей: если в запросах часто встречается какая-либо комбинация условий, то индекс строят именно на эти условия.

Добавление индексов требует дополнительного места на диске, но важнее то, что любые манипуляции с данными становятся более затратными, так как индекс в этом случае должен обновиться. Из-за этого неразумно выстраивать все возможные индексы, следует ограничиться только наиболее востребованными.

Чтобы создать индекс, нужно использовать команду CREATE INDEX, указать имя индекса и список атрибутов, по которым он будет строиться, опционально можно указать тип индекса.

Важно: прежде чем создавать индексы, посмотрите, сколько места занимают ваши данные. Это можно сделать средствами утилиты du или ls -a.

Выполните в psql:

show data\_directory;

чтобы узнать, где хранятся файлы вашей базы. После этого выполните:

SELECT oid from pg\_database where datname = 'imdb';

чтобы узнать идентификатор базы.

Директория с файлами вашей базы находится тут: {data\_directory}/base/{oid} (у меня это: /var/lib/postgresql/9.3/main/base/16384)

Выполните команду du для этого пути (при необходимости прочитайте мануал к команде du).

Пробуйте создавать различные индексы и проверять, как изменяется занимаемое базой место.

**Типы индексов**

Наиболее популярный тип индекса - B-дерево. Эта структура хорошо подходит для атрибутов, которые можно сравнивать между собой, выполняя операции: <, <=, =, >=, >. Также индексы этого типа поддерживают быструю проверку IS (NOT) NULL, поэтому они хорошо подходят для атрибутов с большим количеством пропусков. В-деревья ускоряют любые операции, связанные со сравнением значений, в том числе сортировку и группировку.

Синтаксис создания индекса:

CREATE INDEX index\_name ON table\_name (column\_name);

Второй по популярности тип индекса - hash, который позволяет быстро проводить сравнения на равенство. Этот тип также подходит для условий типа WHERE column\_name IN ( ... ).

Синтаксис создания индекса:

CREATE INDEX index\_name ON table\_name USING HASH (column\_name);

**Функции в индексах**

Удобно для строк создавать также индексы независимые от регистра:

CREATE INDEX index\_name ON table\_name (lower(column\_name));

В этом случае, выбирая WHERE lower(title) = 'the matrix' приведение значений колонки не нужно, и запрос выполнится быстро.

**Индексы на несколько колонок**

Удобный для часто встречающихся комбинаций условий.

Синтаксис:

CREATE INDEX index\_name ON table\_name (column1\_name, column2\_name);

**Уникальный индекс**

Этот индекс также добавляет проверку целостности при изменении данных.

Синтаксис:

CREATE UNIQUE INDEX index\_name on table\_name (column\_name);

**Условные индексы**

Можно создать еще более специализированный индекс на конкретные условия.

CREATE INDEX index\_name on title (kind\_id, production\_year) where (production\_year>2000 and kind\_id=1);

Этот индекс позволит выбирать фильмы выпущенные после 2000 года, при этом для записей не удовлетворяющих условию все останется по-прежнему (как скорость выборки, так и скорость изменения данных).

Внешние ключи

Внешние ключи - это атрибуты, которые связывают сущности, поддерживая ссылочную целостность: нельзя установить значение для foreign key, которого нет в таблице на которую атрибут ссылается. Для внешних ключей СУБД использует hash-индексы.

Пример создания внешнего ключа:

alter table movie\_info add foreign key (movie\_id) references title(id);

Используя схему БД из второй лабораторной работы, добавьте внешние ключи, нужные для ваших запросов.

Запрос для просмотра существующих внешних ключей:

SELECT

tc.constraint\_name, tc.table\_name, kcu.column\_name,

ccu.table\_name AS foreign\_table\_name,

ccu.column\_name AS foreign\_column\_name

FROM

information\_schema.table\_constraints AS tc

JOIN information\_schema.key\_column\_usage AS kcu

ON tc.constraint\_name = kcu.constraint\_name

JOIN information\_schema.constraint\_column\_usage AS ccu

ON ccu.constraint\_name = tc.constraint\_name

WHERE constraint\_type = 'FOREIGN KEY' AND tc.table\_name='mytable';

Пример плана запроса после создания внешнего ключа:

explain

select \* from title t

join movie\_info mi on mi.movie\_id=t.id

where t.production\_year=1899;

План запроса:

"Hash Join (cost=24668.15..680874.12 rows=26044 width=192)"

" Hash Cond: (mi.movie\_id = t.id)"

" -> Seq Scan on movie\_info mi (cost=0.00..247886.02 rows=10198002 width=98)"

" -> Hash (cost=24411.20..24411.20 rows=9436 width=94)"

" -> Bitmap Heap Scan on title t (cost=177.56..24411.20 rows=9436 width=94)"

" Recheck Cond: (production\_year = 1899)"

" -> Bitmap Index Scan on ok (cost=0.00..175.20 rows=9436 width=0)"

" Index Cond: (production\_year = 1899)"

План запроса. EXPLAIN

С помощью команды EXPLAIN можно посмотреть, как предположительно будет выполняться запрос и как он в итоге выполнился. Вывод команды содержит для каждой таблицы, используемой в запросе:

* сколько условных единиц времени уйдет на получение/изменение первой строчки
* сколько условных единиц времени уйдет на выполнение операции целиком
* сколько всего строк будет задействовано

Условные единицы времени связаны с количеством страниц памяти, которое нужно прочитать.

**Основы EXPLAIN**

Пример команды:

explain

select \* from title t

join kind\_type kt on kt.id=t.kind\_id

where production\_year=2000 and kt.id=1;

Ее вывод:

"Nested Loop (cost=0.00..108304.97 rows=15233 width=146)"

" -> Seq Scan on kind\_type kt (cost=0.00..1.09 rows=1 width=52)"

" Filter: (id = 1)"

" -> Seq Scan on title t (cost=0.00..108151.55 rows=15233 width=94)"

" Filter: ((kind\_id = 1) AND (production\_year = 2000))"

Здесь Nested Loop означает, что внутри запроса выполняется цикл. Этот цикл состоит из последовательного просмотра записей через первичный ключ (Seq Scan).

Всего запрос предполагает вернуть примерно 15233 записи.

Если добавить индекс:

create index production\_year\_index on title (production\_year);

То план запроса изменится:

"Nested Loop (cost=1153.79..57625.67 rows=15233 width=146)"

" -> Seq Scan on kind\_type kt (cost=0.00..1.09 rows=1 width=52)"

" Filter: (id = 1)"

" -> Bitmap Heap Scan on title t (cost=1153.79..57472.25 rows=15233 width=94)"

" Recheck Cond: (production\_year = 2000)"

" Filter: (kind\_id = 1)"

" -> Bitmap Index Scan on ok (cost=0.00..1149.98 rows=62073 width=0)"

" Index Cond: (production\_year = 2000)"

Время получения первой строки увеличилось, так как теперь запросу нужно сначала прочитать индексы, а затем выполнять сам запрос. Но и максимальная оценка выполнения запроса уменьшилась вдвое.

Также теперь поиск записей в title происходит не последовательным перебором, а Bitmap Heap Scan - алгоритмом поиска по B-дереву. Именно такого типа индекс был добавлен.

**EXPLAIN ANALIZE**

Опция ANALIZE позволяет сделать более точный прогноз выполнения запроса. Однако, при этом запрос будет частично выполнен, поэтому ее следует использовать в сочетании с ROLLBACK или иметь ввиду, что случайные данные будут обновлены. Также из-за выполнения запроса, эта операция может занять некоторое время (время получения первых строк результата).

ANALIZE показывает точную оценку времени выполнения запроса в миллисекундах. И количество прохождений циклов.

Пример запроса:

explain ANALYZE

select \* from title t where production\_year=2000 and kind\_id in (select id from kind\_type where id=1);

Пример вывода:

"Nested Loop (cost=1153.79..57625.67 rows=15233 width=94) (actual time=234.085..339.698 rows=8763 loops=1)"

" -> Seq Scan on kind\_type (cost=0.00..1.09 rows=1 width=4) (actual time=0.006..0.009 rows=1 loops=1)"

" Filter: (id = 1)"

" Rows Removed by Filter: 6"

" -> Bitmap Heap Scan on title t (cost=1153.79..57472.25 rows=15233 width=94) (actual time=234.076..338.145 rows=8763 loops=1)"

" Recheck Cond: (production\_year = 2000)"

" Rows Removed by Index Recheck: 1795048"

" Filter: (kind\_id = 1)"

" Rows Removed by Filter: 52237"

" -> Bitmap Index Scan on ok (cost=0.00..1149.98 rows=62073 width=0) (actual time=18.910..18.910 rows=61000 loops=1)"

" Index Cond: (production\_year = 2000)"

"Total runtime: 340.103 ms"

**EXPLAIN (ANALYZE, BUFFERS)**

Опция BUFFERS показывает использование буфера во время выполнения запроса. Значение shared hit показывает, сколько блоков буфера удалось использовать, то есть избежать повторного чтения из основном памяти, shared read - сколько блоков считалось из таблиц и индексов.

Дополнительно

* [Подробнее про работу индексов с примерами (спойлеры)](http://wiki.cs.hse.ru/%D0%91%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85/%D0%9B%D0%B0%D0%B1%D0%BE%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D0%B0%D1%8F_%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0_3/%D0%98%D0%BD%D0%B4%D0%B5%D0%BA%D1%81%D1%8B)
* <http://www.postgresql.org/docs/9.3/static/sql-explain.html>
* <https://habrahabr.ru/post/203320/>
* <https://habrahabr.ru/post/203386/>
* <https://habrahabr.ru/post/203484/>

Параметры и состояние PostgreSQL-сервера

В этом разделе собраны инструменты и техники для мониторинга и оценки статуса сервера СУБД и сбора статистики.

Для мониторинга текущих запросов разработана утилита аналогичная программе top - [pg\_activity](https://github.com/julmon/pg_activity). Эта программа показывает текущие запросы и используемые ресурсы.

Просмотри статистики выполнения операций в базе:

select \* from pg\_stat\_database where datname='imdb';

Некоторые значения в статистике:

* xact\_commit - количество закоммиченных транзакций (в том числе и обычных операций insert/update)
* xact\_rollback - количество сроллбеченных транзакций (в том числе и обычных операций insert/update)
* blks\_read - общее количество блоков, которые были считаны
* blks\_hit - количество блоков, считанных из кэша
* tup\_returned - количество кортежей, которые были выведены для пользователя
* tup\_fetched - количество кортежей, которые СУБД выбирала для последующих операций (сравните со значением выше)
* temp\_bytes - размер временных файлов, которые создаются для ускорения выборки (например, при присоединении таблиц)
* deadlocks - количество транзакций с взаимной блокировкой

Попробуйте выполнить запрос, сравнив количество считываемых блоков в его плане выполнения и изменениями статистики.

Есть еще несколько таблиц со статистикой по использованию памяти для таблиц и индексов. Единственный интерес в них представляют количества live/dead tuples. Мертвые кортежи - это те, которые уже удалены из базы логически, но на их место в памяти еще не вставлены новые.

Дополнительно:

* <http://www.postgresql.org/docs/current/static/monitoring-stats.html>
* <https://github.com/julmon/pg_activity>