**Лабораторные работы по курсу**

**Базы данных**

**Лабораторная работа 7**

**«Индексы, транзакции»**

**Москва, 2024**

Оглавление

[1. Теоретическая часть 3](#_Toc165222157)

[1.1. Индексы 3](#_Toc165222158)

[1.2. Анализ выполнения запросов 4](#_Toc165222159)

[1.3. Транзакции 4](#_Toc165222160)

[2. Практическая часть 7](#_Toc165222161)

[2.1. Задание 1. 7](#_Toc165222162)

[2.2. Задание 2. 9](#_Toc165222163)

[2.3. Задание 3. 10](#_Toc165222164)

[Контрольные вопросы 10](#_Toc165222165)

[Список использованной литературы 10](#_Toc165222166)

1. Теоретическая часть

В предыдущих лабораторных работах было показано как спроектировать БД, создать таблицы и заполнить их требуемой информацией. В данной работе рассмотрим средства, оптимизирующие и ускоряющие работу с таблицами – введем понятия индекса и транзакции.

* 1. Индексы

При работе с базами данных очень часто необходимо выполнять задачи, связанные с поиском строк в таблицах. Для ускорения подобных запросов используются индексы.

Индекс – специальная структура данных, которая связана с таблицей и создается на основе1 данных, содержащихся в ней. Основная цель создания индексов – повышение производительности функционирования базы данных. [1]

В общем случае индексы представляют собой дополнительную структуру, содержащую значение индексируемого атрибута и указатель на данный элемент. Все записи в индексе являются упорядоченными, поэтому поиск данных значительно ускоряется.

В качестве примера можно привести алфавитный указатель, расположенный в конце книги. Обычно, указатель упорядочен по алфавиту и помимо списка определений содержит страницы, на которых располагается о них информация. Когда мы хотим найти значение некоторого слова, то обращаемся к указателю, быстро находим требуемое значение и переходим на страницу, указанную рядом с ним. Таким образом, отпадает необходимость последовательного просмотра всех страниц в книге.

В данном примере книга – это база данных, индекс – алфавитный указатель.

На практике индексы применяются, когда объемы базы данных существенно велики. Недостатком применения индексов является то, что они занимают отдельное место не диске и требуют накладных расходов для поддержания их в актуальном состоянии при выполнении обновления таблицы.

В предыдущих работах рассматривалось задание первичного ключа PRIMARY KEY. При его реализации СУБД сама автоматически создает индекс. Однако, при разработке БД очень часто требуется создавать дополнительные индексы, с учетом наиболее частых запросов.

Создание индекса происходит с помощью следующей команды, записанной в сокращенном виде:

CREATE [ UNIQUE ] INDEX *имя\_индекса* ON *имя\_таблицы*

( { *имя\_столбца* | ( *выражение* ) }

[ ASC | DESC ] [ NULLS { FIRST | LAST } ] [, ...] )

Параметр UNIQUE отслеживает уникальность при создании индекса и контролирует добавление или изменение записи, выдавая ошибку при появлении повторяющихся значений. Указание *имя\_индекса* необязательно, PostgreSQL формирует подходящее имя по имени родительской таблицы и именам индексируемых столбцов.

При создании индекса обязательно указывается ***имя столбца*** или ***выражение***с одним или несколькими столбцами, по которому строится индекс. Параметр ***ASC*** указывает порядок сортировки по возрастанию (подразумевается по умолчанию), ***DESC*** - по убыванию, ***NULLS FIRST/*** ***LAST*** указывает, что значения NULL после сортировки оказываются перед/после остальных.

Например, создание индекса для атрибута «Номер группы» таблицы Студент происходит следующим образом.

CREATE INDEX id\_index ON student (students\_group\_number);

Таким образом, при больших объемах базы данных, запросы, содержащие ключевое слово WHERE, будут выполняться быстрее.

* 1. Анализ выполнения запросов

Для разработки эффективного SQL запроса необходимо учитывать множество факторов, влияющих на его работу – применяемые планировщиком запросов алгоритмы, количество строк в таблице, используемые типы данных и т. п. Для того, чтобы получить поэтапный алгоритм выполнения запроса, составленный СУБД возможно провести анализ запроса и вывести план его выполнения. В PostgreSQL для этого существуют ключевые слова EXPLAIN ANALYZE. Например, с помощью команды

EXPLAIN ANALYZE SELECT \* FROM students;

на экран выводится следующий план выполнения запроса:

QUERY PLAN

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

Seq Scan on students (cost=0.00..12.83 rows=483 width=87) (actual time=0.010..0.036 rows=483 loops=1)

Planning Time: 0.051 ms

Execution Time: 0.053 ms

(3 rows)

Планы запросов анализируются снизу вверх. Реальное время выполнения данного запроса – 0,053 мс. По алгоритмам Postgres время должно было составить 0,051 мс.

Более интересна первая строка. В ней указан алгоритм, по которому проводилось чтение строк – *seq scan* – последовательный поиск. Далее указывается таблица, с которой ведется работа и в скобках анализируемые параметры. Cost – «стоимость», численная оценка предполагаемого количества требуемых ресурсов для вычисления запроса. Первая оценка (0.00) – число ресурсов для вывода первой строки и последняя (12.83) – для вывода всего массива данных. Значение стоимостей берется из анализа накопленной статистики, значений сложности вычисления используемых алгоритмов и системных настроек [2]. Далее указывается приблизительное число строк, которое будет выведено в результате запроса (483) и приблизительная их средний размер. Во вторых круглых скобках аналогично указывается предположительное время работы.

При усложнении запроса усложняется и план выполнения.

Рассмотрим запрос, в котором наложено условие фильтрации по полю с именем студента.

EXPLAIN ANALYZE SELECT \* FROM students

WHERE first\_name='Иван';

QUERY PLAN

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

Seq Scan on students (cost=0.00..14.04 rows=1 width=87) (actual time=0.047..0.047 rows=0 loops=1)

Filter: ((first\_name)::text = 'Иван'::text)

Rows Removed by Filter: 483

Planning Time: 0.075 ms

Execution Time: 0.057 ms

(5 rows)

Аналогично, анализируем план снизу-вверх. Первым делом была применена операция фильтрации и с её помощью было удалено из потенциально выводимых значений 483 строки. Далее происходит последовательная выборка оставшихся строк и их пересылка клиенту. Обратите внимание, что в текущем варианте, значение rows=1, хотя на самом деле, число вернувшихся строк должно быть равно 25. Это связано с тем, что на текущий момент еще не накопилась статистика в СУБД по выполнению данного запроса и оптимизатор допустил ошибку. Для того, чтобы избежать этой ошибки, нужно еще несколько раз запустить запрос.

* 1. Транзакции

При работе с базами данных часто требуется выполнять несколько запросов последовательно. Например, при добавлении информации о студенте в учебную базу данных необходимо произвести две записи с помощью запроса INSERT – вставку информации в таблицу *students* и связанную с ней *student\_ids*. В случае, если произойдет сбой при выполнении одного из запросов, а второй выполнится, то в базе данных будет храниться лишь частичная информация о студенте. Для борьбы с подобными ошибками используются **транзакции**.

Транзакции – совокупность операций над базой данных, которые вместе образуют логически целостную процедуру, и могут быть либо выполнены все вместе, либо не будет выполнена ни одна из них. [1]

Для корректности работы системы транзакции должны обладать следующими свойствами:

1. Атомарность (**A**tomicity) – все операции внутри транзакции гарантированно должны выполниться или не выполниться ни одна из них.
2. Согласованность (**C**onsistency) ­– база данных в результате выполнения транзакции переходит из одного согласованного состояния в другое
3. Изолированность (**I**solation) – во время выполнения транзакции, другие транзакции должны по возможности минимально влиять на её ход работы
4. Долговечность (**D**urability) – после завершения работы транзакции данные должны быть надежно сохранены в базе данных

Еще одно применение транзакций – параллельная работа с базой данных нескольких пользователей. Если они одновременно попытаются изменить одну и ту же запись, то будет неизвестно, какое изменение в итоге сохранится. Попытка же заблокировать запись при вводе для остальных пользователей и организация последовательного выполнения транзакций, приведет к сильной потери производительности системы. Поэтому в СУБД PostgreSQL реализованы параллельно исполняемые транзакции. Реализация транзакций основана на многоверсионной модели (MVCC), которая предполагает, что каждый пользователь видит так называемый «снимок» данных, то есть согласованную версию БД, которую она имела на определенный момент времени. Когда параллельные транзакции вносят изменения в одну и ту же строку, на самом деле создаются отдельные версии этих строк, доступных определенной транзакции. Для конкретных задач может требоваться различная степень независимости параллельных транзакций. Поэтому, существует понятие уровня изоляции транзакции. Каждый уровень характеризуется перечнем не допустимых ошибок при параллельных транзакциях.

Рассмотрим возможные ошибки, которые могут возникнуть при параллельном выполнении транзакций:

1. Потерянное обновление. Представим ситуацию, когда две транзакции одновременно изменяют одни и те же данные. В итоге сохранено будет только одно из двух значений, без учета изменений второй транзакции.
2. «Грязное» чтение. Первая транзакция прочитала данные, которые только что изменила и еще не зафиксировала вторая транзакция. Если в итоге если вторая транзакция была отменена, то первой были прочитаны некорректные значения.
3. Неповторяющееся чтение. В ходе одной транзакции происходит два чтения одних и тех же данных. Между ними происходит запись и фиксация этих же данных другой транзакцией. Таким образом, первая транзакция прочитает два разных значения.
4. Фантомное чтение. В ходе первой транзакции происходит выборка строк из таблицы. Вторая транзакция производит изменения значений в данной таблице. Таким образом, если в рамках первой транзакции снова будет выполнен запрос на выборку, то результат будет отличаться от первого.
5. Аномалия сериализации. В данном случае результат выполнения нескольких транзакций последовательно (при любом порядке их следования) не будет совпадать с зафиксированным результатом параллельного их выполнения.

В стандарте SQL предусматривается четыре уровня изоляции. Для простоты сведем их в таблицу.

Таблица 1 Уровни изоляции транзакций

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Уровень изоляции | «Грязное» чтение | Неповторяемое чтение | Фантомное чтение | Аномалия сериализации |
| Read uncommited | + (в PosgreSQL НЕ допускается) | + | + | + |
| Read committed | - | + | + | + |
| Repeatable read | - | - | + (в PosgreSQL НЕ допускается) | + |
| Serializable | - | - | - | - |

Обратите внимание, что в PosgreSQL некоторые уровни изоляции устроены несколько строже, чем указано в стандарте языка SQL. Таким образом, уровни Read committed и Read uncommitted в PosgreSQL совпадают.

По умолчанию, PosgreSQL использует уровень изоляции Read committed. Для просмотра текущего уровня изоляции в PosgreSQL используется команда

SHOW transaction\_isolation;

В PosgreSQL транзакция определяется набором SQL-команд, окружённым командами BEGIN и COMMIT.

BEGIN;

-- ...

COMMIT;

Если в процессе выполнения транзакции мы решим, что не хотим фиксировать её изменения, то возможно выполнить команду ROLLBACK вместо COMMIT, чтобы все наши изменения были отменены. Для задания конкретного уровня изоляции транзакции необходимо использовать конструкцию

BEGIN ISOLATION LEVEL *название уровня изоляции*

Например,

BEGIN ISOLATION LEVEL REPEATABLE READ;

Рассмотрим следующий пример:

Запустим на выполнение транзакцию и выведем содержимое атрибута «Ставка» из таблицы «Трудоустройство» на экран.

BEGIN;

SELECT wage\_rate FROM employment;

wage\_rate

-----------

0.25

0.75

0.15

0.50

0.50

1.00

1.50

0.25

Обновим значение ставки и выведем новые значения на экран.

UPDATE employment

SET wage\_rate = wage\_rate \* 1.10;

SELECT wage\_rate FROM employment;

wage\_rate

-----------

0.28

0.83

0.17

0.55

0.55

1.10

1.65

0.28

Откроем еще один экземпляр Query tool и выполним из-под нового клиента запрос на вывод ставок.

SELECT wage\_rate FROM employment;

wage\_rate

-----------

0.25

0.75

0.15

0.50

0.50

1.00

1.50

0.25

Обратите внимание, что т.к. транзакция не была завершена, то значения ставок еще не изменились. Для её завершения выполним команду COMMIT от первого клиента

COMMIT

Повторим запрос со второго клиента, и убедимся, что число ставок обновилось.

wage\_rate

-----------

0.28

0.83

0.17

0.55

0.55

1.10

1.65

0.28

Вернем обратно значения ставок. Для этого выполним запрос:

UPDATE employment

SET wage\_rate = wage\_rate / 1.10;

wage\_rate

-----------

0.25

0.75

0.15

0.50

0.50

1.00

1.50

0.25

Повторим предыдущую транзакцию, только вместе команды COMMIT выполним команду ROLLBACK.

BEGIN;

UPDATE employment

SET wage\_rate = wage\_rate \* 1.10;

ROLLBACK;

Убедимся в том, что значения не были изменены в результате транзакции.

1. Практическая часть
   1. Задание 1.

Для глубокого анализа данного задания вам необходимо ознакомиться с рекомендованной литературой, указанной в источниках.

* + 1. *Исследование производительности системы*

Создайте таблицу, содержащую значения посещаемости студентом института. Таблица содержит номер студенческого билета, время его входа, выхода и сгенерированное случайное кодовое число при выходе из вуза.

**CREATE TABLE attendance (**

**attendance\_id SERIAL PRIMARY KEY,**

**generated\_code VARCHAR(64),**

**person\_id integer,**

**enter\_time timestamp,**

**exit\_time timestamp,**

**FOREIGN KEY (person\_id) REFERENCES student\_ids (student\_id)**

**);**

С помощью следующего скрипта заполните таблицу данными.

**do**

**$$**

**DECLARE**

**enter\_time timestamp(0);**

**exit\_time timestamp(0);**

**person\_id integer;**

**enter\_id VARCHAR(64);**

**BEGIN**

**FOR i IN 1..1000000 LOOP**

**-- Генерируем случайную дату в указанном диапазоне**

**enter\_time := to\_timestamp(random() \***

**(**

**extract(epoch from '2023-12-31'::date) -**

**extract(epoch from '2023-01-01'::date)**

**)**

**+ extract(epoch from '2023-01-01'::date)**

**);**

**-- Генерируем случайный интервал времени, который пробыл в вузе студент (не более 10 часов)**

**exit\_time := enter\_time + (floor(random() \* 36000 + 1)\*'1 SECOND'::interval);**

**person\_id := (**

**SELECT student\_id FROM students**

**ORDER BY random()**

**LIMIT 1**

**);**

**enter\_id := md5(random()::text);**

**INSERT INTO attendance(generated\_code, person\_id, enter\_time,exit\_time)**

**VALUES(enter\_id, person\_id, enter\_time, exit\_time);**

**END LOOP;**

**END**

**$$;**

Добавьте в таблицу *attendance*одно значение, измерив время данной операции. Далее измерьте время выполнения запроса, выводящего содержимого таблицы в отсортированном виде по столбцу *generated\_code*.

Добавьте индекс на столбец *generated\_code*. Повторите предыдущие две операции. Сравните полученное время. Во сколько раз оно изменилось? Результаты вычисления занесите в таблицу.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Время до индексирования Tb | Время после индексирования Ta | Ta/ Tb |
| SELECT |  |  |  |
| INSERT |  |  |  |

* + 1. *Индексы и селективность*

Выполните запрос, выводящий все строки таблицы *attendance,* измерьте время его выполнения. Добавьте условие, выбрав только все записи, связанные с одним конкретным студентом. Аналогично измерьте время выполнения. Создайте индекс на атрибут *person\_id* и повторите эксперименты. Сравните время выполнения операций до создания индекса и после. Объясните полученный результат.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Время до индексирования Tb | Время после индексирования Ta | Ta/ Tb |
| SELECT |  |  |  |
| SELECT + WHERE |  |  |  |

* + 1. *Анализ плана выполнения запроса*

Составьте запрос к таблице *attendance*, выводящий все строки в отсортированном порядке, в которых столбец *generated\_code* заканчивается символом ‘*a*’. Проанализируйте полученный запрос и объясните результат. Используется ли в данном случае индекс?

* 1. Задание 2.

Предположим, что студент группы ИВТ-42 Полиграф Шариков во время зимней сессии пересдал экзамен по дисциплине «Операционные системы» на оценку 5 и пересдал экзамен по дисциплине «Базы данных» на 5. Одновременно с проставлением баллов за его успехами следила методист кафедры. Для работы с несколькими транзакциями запустите два командных окна (запросника). В первом вводите команды за преподавателя, проставляющего оценки, а во втором за методиста, просматривающего результаты.

* + 1. *Работа с транзакциями*

В рамках транзакции измените значение оценки студента по Операционным системам и проверьте значение в первом и втором окне. Зафиксируйте изменения и вновь проверьте значения. Аналогично внесите новую оценку по Базам данных и проверьте изменения.

|  |  |
| --- | --- |
| Преподаватель | Методист |
| BEGIN;  *Изменяет оценку*  *Добавляет оценку*  COMMIT | *Смотрит результат до фиксации изменений преподавателем*  *Смотрит результат после фиксации изменений преподавателем* |

* + 1. *Отмена изменений транзакций*

Удалите добавленное значение и верните исправленную оценку в прежнее состояние. Повторите аналогичные действия, только по окончании внесения изменений преподавателем откатите их с помощью команды ROLLBACK. Какое значение увидела методист?

* + 1. *Моделирование аномалий при выполнении транзакций*

Повторите эксперименты в п. 2.3.1, используя различные уровни изоляции.

|  |  |
| --- | --- |
| Преподаватель | Методист |
| BEGIN ISOLATION LEVEL …  *Изменяет оценку*  *Добавляет оценку*  COMMIT | BEGIN ISOLATION LEVEL …  *Смотрит результат до фиксации изменений преподавателем*  *Смотрит результат после фиксации изменений преподавателем* |

Внесите в таблицу в какой момент были получены ошибочные значения из-за аномалий.

|  |  |  |
| --- | --- | --- |
| Уровень изоляции | До фиксации | После фиксации |
| Read uncommited |  |  |
| Read committed |  |  |
| Repeatable read |  |  |
| Serializable |  |  |

Как вы считаете, какой уровень изоляции необходимо использовать на практике и почему?

* 1. Задание 3.

Проанализируйте учебную базу данных и проиндексируйте одно из полей любой таблицы. Объясните свой выбор.

Контрольные вопросы

1. За счет чего индексы ускоряют выборку данных?
2. Существуют ли случаи, когда использование индексов замедляет выборку данных?
3. Какая структура данных хранит в себе индексные записи?
4. Для чего предназначены транзакции?
5. В чем отличие между неповторяющимся и фантомным чтением?

Список использованной литературы

|  |  |
| --- | --- |
| [1] | Е. П. Моргунов, PostgreSQL. Основы языка SQL, 1-е ред., Санкт-Петербург: БХВ-Петербург, 2018, p. 336. |
| [2] | Г. Домбровская, Б. Новиков и А. Бейликова, Оптимизация запросов в PostgreSQL, Москва: ДМА, 2022. |
| [3] | «Исходный код СУБД postgres,» [В Интернете]. Available: https://github.com/postgres/postgres. [Дата обращения: 30 01 2023]. |
| [4] | Документация к PostgreSQL 15.1, 2022. |
| [5] | Е. Рогов, PostgreSQL изнутри, 1-е ред., Москва: ДМК Пресс, 2023, p. 662 . |
| [6] | Б. А. Новиков, Е. А. Горшкова и Н. Г. Графеева, Основы технологии баз данных, 2-е ред., Москва: ДМК пресс, 2020, p. 582. |
|  |  |