Лабораторные работы по курсу
Базы данных

Лабораторная работа 6
«Индексы, транзакции, представления»

Москва, 2023

Оглавление

1. Теоретическая часть

В предыдущей лабораторной работе рассматривались вопросы, связанные с проектированием базы данных. В данной лабораторной работе предложены темы, связанные с изменением данных в таблицах – вставкой, обновлением и удалением. В конце работы рассмотрено понятие транзакции в базе данных.

1.1. 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 основе 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.

Создание индекса происходит с помощью следующей команды:

```
CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ] [ [ IF NOT EXISTS ] имя_индекса ] ON имя_таблицы [ USING метод ]
    ( { имя_столбца | ( выражение ) } [ COLLATE правило_сортировки ] [ класс_операторов ] [ ASC | DESC ] [ NULLS { FIRST | LAST } ] [, ...] )
```

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.

1.2. 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 – на вставку информации в таблицу *student* и *student_id*. 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 вместе одновременно попытаются изменить значение одного из значений, то будет неизвестно, какое из них в итоге сохранится. Однако таким образом, если транзакции не смогут выполняться одновременно и будут строго идти одна за другой, то это сильно повлияет на производительность всей системы. Поэтому, существует понятие уровня изоляции транзакции. Каждый уровень характеризуется наличием возможных типов ошибок, которые могут произойти при некорректной работе.

Рассмотрим подобные 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
2. «Грязное» чтение. Первая транзакция прочитала данные, которые только что изменила вторая транзакция. Если в итоге 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 параллельным их выполнением.

Для определения, какие из приведенных выше ошибок возможны в рамках выполнения транзакции, а какие нет, существует понятие уровня изоляции. В стандарте SQL предусматривается четыре таких уровня. 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.

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.

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

```
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.

1.3. Представления

При работе с базами данных очень часто приходится выполнять одинаковые сложные и объемные запросы. Для упрощения работы возможно сформировать из такого запроса представление, к которому далее возможно обращаться, как будто это обычная таблица.

Например, создадим представление на основе запроса, выводящего трудоемкость дисциплины в формате «Количество часов/ЗЕТ»

```
CREATE VIEW labor_intensity AS
SELECT field_name AS "Field name", (36*zet::numeric)::varchar || '/' || zet
as "Labor intensity"
FROM field
ORDER BY "Labor intensity"
```

Далее к созданному представлению возможно обратиться, как к таблице

```
SELECT * FROM labor_intensity
```

```
                          Field name                          | Labor intensity
--------------------------------------------------------------+-----------------
 Риторика                                                     | 108/3
 Основы управления проектами                                  | 108/3
 Основы рыночной экономики                                    | 108/3
 Безопасность жизнедеятельности                               | 108/3
 Экология                                                     | 108/3
 Иностранный язык                                             | 108/3
 История                                                      | 108/3
 Правоведение                                                 | 108/3
```

2. Практическая часть

2.1. Задание 1.

Проведите следующий эксперимент:

Создайте простую таблицу, содержащую три поля – суррогатный ключ и два атрибута, содержащие строки.

```
CREATE TABLE Test(
      id SERIAL PRIMARY KEY,
      CODE_1 VARCHAR(64),
      CODE_2 VARCHAR(64)
);
```

С помощью следующего скрипта заполните таблицу данными.

```
DO
$$
BEGIN
FOR i IN 1..1000000 LOOP
    INSERT INTO Test(code_1,code_2) VALUES(md5(random()::text),
md5(random()::text));
END LOOP;
END
$$ language plpgsql;
```

Обратите внимание, что данный скрипт написан на процедурном языке PL/pgSQL. Более подробно о нем можно прочитать в 7 лабораторной работе. [2]

С помощью функций EXPLAIN ANALYZE в PostgreSQL возможно производить анализ запросов и вычислять затрачиваемое время на его выполнение. Например,

```
EXPLAIN ANALYZE SELECT * FROM student;
```

Добавьте в таблицу *Test* одно значение, измерив время данной операции. Далее измерьте время выполнения запроса, выводящего содержимого таблицы в отсортированном виде по столбцу CODE_1.

Добавьте индекс на столбец CODE_1. Повторите предыдущие две операции. Сравните полученное время. Во сколько раз оно изменилось? Результаты вычисления занесите в таблицу.

2.2. Задание 2.

Составьте запрос к таблице Test, выводящий все строки в отсортированном порядке, в которых столбец CODE_1 начинается с символа ‘a’. Проанализируйте полученный запрос и объясните результат.

2.3. Задание 3.

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

2.4. Задание 4.

Предположим, преподаватель зашел в электронный журнал, чтобы проставить студенту оценку за дисциплину. Одновременно студент решил проверить, выставлена ли ему оценка. Смоделируйте данную ситуацию, заключив действия студента и преподавателя в транзакции с уровнем изоляции, в соответствии с вариантом. Приведите пример аномалий, которые могут возникнуть.

Вариант	Уровень изоляции
1	Read uncommited
2	Read committed
3	Repeatable read

4	Serializable
5	Read uncommited
6	Read committed
7	Repeatable read
8	Serializable
9	Read committed
10	Repeatable read

2.5. Задание 5.

Создайте на основе любых запросов из предыдущих лабораторных работ составьте 2–3 представление. Объясните свой выбор.

Список литературы

[1] Е. П. Моргунов, PostgreSQL. Основы языка SQL, 1-е ред., Санкт-Петербург: БХВ-Петербург, 2018, p. 336.

[2] «PL/pgSQL — процедурный язык SQL,» [В Интернете]. Available: https://postgrespro.ru/docs/postgresql/15/plpgsql. [Дата обращения: 09 03 2023].

[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.