Практика 3

Физическая организация данных таблицы

Транзакции, уровни изоляции транзакций

Цель практики:

1. Посмотреть, как физически организована запись в таблице
2. Научиться разбираться в уровнях изоляции транзакции

Задание:

1. Чем отличаются типы данных char и varchar в части их хранения в странице таблицы? Какой тип выгоднее использовать и почему?

   Дополнительное задание (необязательное): посмотреть, как хранятся разные типы данных в странице (для этого нужно создать новые таблицы с другими типами данных)

2. Узнать текущий уровень изоляции транзакций. Как вы его определили?

   (Подсказка: для этого нужно использовать параллельные транзакции)

3. Меняя уровни транзакций, посмотреть, какие проблемы параллельного доступа возможны на каждом уровне.

Теория к заданию 1

Инструмент в PostgreSQL, которые помогают посмотреть содержимое страницы: pageinspect

Как его включить и использовать?

- зайти под пользователем с админскими правами (postgres)
- выполнить команду для включения дополнительного модуля

  CREATE EXTENSION pageinspect;

- теперь будут доступны функции для исследования страниц БД

  https://postgrespro.ru/docs/postgresql/12/pageinspect

- содержимое страницы можно посмотреть командой

  SELECT * FROM heap_page_items(get_raw_page('test',0));

  test - название таблицы,

  0 - номер страницы, информация о которой будет выводиться

Что можно увидеть в результате?

(пример вывода)

```
lp | lp_off | lp_flags | lp_len | t_xmin  | t_xmax | t_field3 | t_ctid | t_infomask2 | t_infomask | t_hoff | t_bits | t_oid |                               t_data
----+--------+----------+--------+---------+--------+----------+--------+-------------+------------+--------+--------+-------+--------------------------------------------------------------------
  1 |   8136 |        1 |     51 | 3716857 |      0 |        0 | (0,1)  |           2 |       2050 |     24 |        |       | \x010000002f746578742077697468207366d652073796d626f6c73
  2 |   8096 |        1 |     33 | 3716858 |      0 |        0 | (0,2)  |           2 |       2050 |     24 |        |       | \x020000000b74657874
  3 |   8032 |        1 |     58 | 3716859 |      0 |        0 | (0,3)  |           2 |       2050 |     24 |        |       | \x030000003d6d6179626576657279 6c6f6e6774657874206f72206d617962656e6f74
  4 |   8000 |        1 |     30 | 3716860 |      0 |        0 | (0,4)  |           2 |       2050 |     24 |        |       | \x040000000571
(4 rows)
```

lp	line pointer - номер указателя на запись внутри страницы
lp_off	смещение до кортежа от начала страницы (в байтах)
lp_flags	состояние line poiner (0 - не используется, 1 - используется, 3 - HOT redirect, 3 - dead)
lp_len	размер кортежа в байтах
t_xmin	номер транзакции, которая добавила данные
t_xmax	номер транзакции, которая удалила данные
t_field3	id команды, которая добавила или удалила данные, или id команды vacuum full
t_ctid	id кортежа (номер страницы, номер кортежа в странице)
t_infomask2	различные флаги*
t_infomask	различные флаги*
t_hoff	размер заголовка строки
t_bits	bitmap of NULLs (указание, что в поле существует NULL значение)
t_oid	
t_data	данные (сами данные или указатели на них)

* при желании можно посмотреть здесь

https://github.com/postgres/postgres/blob/master/src/include/access/htup_details.h

Подготовка к заданию 1

1. создать две таблицы, в одной поле типа char(N), в другой - varchar(N).
2. заполнить таблицы одинаковыми данными, 5-10 строк разной длины будет достаточно. Добавляемые строки должны быть разной длины. Пример заполненных таблиц:

```
student=# select * from only_char;
                                char_string
-------------------------------------------------------------------
 the transaction which created the tuple
 the transaction which deleted or updated the tuple. otherwise 0
 t_xmin
 q
(4 rows)

student=# select * from only_varchar;
                          char_string
-------------------------------------------------------------
 the transaction which created the tuple
 the transaction which deleted or updated the tuple. otherwise 0
 t_xmin
 q
(4 rows)
```

3. используя команду

   SELECT * FROM heap_page_items(get_raw_page('test',0)); (здесь нужно использовать нужные данные)

   посмотреть, как устроена страница таблицы

Теория к заданиям 2-3

Пример транзакции:

```
BEGIN;
UPDATE accounts SET balance = balance -
100.00 WHERE name = 'Alice';
UPDATE accounts SET balance = balance +
100.00 WHERE name = 'Bob';
COMMIT;
```

COMMIT - подтверждение транзакции
ROLLBACK - откат транзакции

В дополнение к обычным данным таблицы можно посмотреть метаданные. Пример запроса:

```
SELECT ctid, xmin, xmax, * FROM test;
```

Метаданные в строке:

- ctid - физическое расположение строки в таблице (номер страницы, номер кортежа в странице)
- xmin - id транзакции, добавившей эту строку
- xmax - id транзакции, удалившей строку, или 0 для неудалённой версии строки. Значение ненулевым и для видимой версии строки. Это означает, что удаляющая транзакция ещё не была зафиксирована, или удаление было отменено
- cmin - номер команды (начиная с нуля) внутри транзакции, добавившей строку

- cmax - номер команды в удаляющей транзакции или ноль

* на самом деле не только они, есть ещё разные флаги и т. д., их можно было увидеть в задании 1

Уровни изоляции транзакций (подробнее в слайдах лекции 3):

1. Read uncommited (Чтение незафиксированных данных)
2. Read committed (Чтение зафиксированных данных)
3. Repeatable read (Повторяемое чтение)
4. Serializable (Сериализуемость)

Проблемы параллельного доступа:

1. «грязное» чтение: транзакция читает данные, записанные параллельной незавершённой транзакцией.
2. неповторяемое чтение: транзакция повторно читает те же данные, что и раньше, и обнаруживает, что они были изменены другой транзакцией (которая завершилась после первого чтения).
3. фантомное чтение: транзакция повторно выполняет запрос, возвращающий набор строк для некоторого условия, и обнаруживает, что набор строк, удовлетворяющих условию, изменился из-за транзакции, завершившейся за это время.
4. аномалия сериализации: результат успешной фиксации группы транзакций оказывается несогласованным при всевозможных вариантах исполнения этих транзакций по очереди

Изменение уровня изоляции транзакции

SET TRANSACTION **режим_транзакции**

или

SET SESSION CHARACTERISTICS AS TRANSACTION **режим_транзакции**

где **режим_транзакции =** ISOLATION LEVEL { READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE }

Пример:

```
SET SESSION CHARACTERISTICS AS TRANSACTION ISOLATION LEVEL REPEATABLE READ;
```

- Команда SET TRANSACTION устанавливает характеристики текущей транзакции. На последующие транзакции она не влияет
- SET SESSION CHARACTERISTICS устанавливает характеристики транзакции по умолчанию для последующих транзакций в рамках сеанса. Заданные по умолчанию характеристики затем можно переопределить для отдельных транзакций командой SET TRANSACTION.

https://postgrespro.ru/docs/postgrespro/12/sql-set-transaction