МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский университет ИТМО»

ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ

Курсовая работа

по дисциплине

‘Информационные системы и базы данных’

Этап №3

Выполнили:
Студенты группы P33312
Соболев Иван
Александрович,
Кизилов Степан
Александрович
Преподаватель:
Наумова Надежда
Александровна

Санкт-Петербург, 2023

Задание:
Реализовать даталогическую модель в реляционной СУБД PostgreSQL:
• Создать необходимые объекты базы данных
• Заполнить созданные таблицы тестовыми данными
• Сделать скрипты для:
создания/удаления объектов базы данных
заполнения/удаления созданных таблиц
• Обеспечить целостность данных при помощи средств языка DDL.
• Добавить в базу данных триггеры для обеспечения комплексных
ограничений целостности
• Реализовать функции и процедуры на основе описания бизнес-процессов (из этапа №1)
• Произвести анализ использования созданной базы данных:
выявить наиболее часто используемые запросы к объектам базы
данных
результаты представить в виде текстового описания
• Создать индексы и доказать, что они полезны для вашей базы данных:
доказательство должно быть приведено в виде текстового
описания

Выполнение:

Скрипты запросов:
Ivanio1/itmo-isbd-coursework (github.com)

Триггеры:
Создали триггеры для обеспечения целостности данных:

update_tool_stock_on_purchase_update_trigger – позволяет автоматически регулировать количество свободного инструмента на основе состояния заказов.

update_detail_stock_on_purchase_update_trigger - позволяет автоматически регулировать количество деталей на основе состояния заказов

update_purchase_closed_at_trigger - позволяет автоматически регулировать время закрытия заказов

check_unique_email_trigger – выдаёт ошибку, если происходит добавление пользователя с уже имеющимся в базе данных email

Функции:
Реализовали функции на основе главных бизнес-процессов:

create_purchase - Создать заказ, автоматически заполнив её состояние и дату создания

update_purchase_status_to_done, update_purchase_status_to_waiting,
update_purchase_status_to_in_process – изменяют состояние заказа

get_available_tools – возвращает список инструментов, которые в данный момент свободны
get_zero_tools - Возвращает инструменты, которые закончились (/сломались)

fill_tool_count, fill_tool_count_by_name – при покупке инструментов увеличить их количество

is_stock_of_details_greater, is_stock_of_details_greater_by_name – проверить количество доступных деталей

fill_detail_count, fill_detail_count_by_name – при поступлении деталей увеличить их количество на складе

Анализ:
Наиболее часто используемая информация храниться в таблицах Purchase, Car, Offer, Client, Detail, Tool, STO.
В таблице Purchase основным полем является статус заказа (state). К данному полю обращаются клиенты, когда отсматривают на каком этапе находится заказ, администраторы, которые следят за тем, чтобы работники выполняли заказы в срок, а также с этим полем производят манипуляции и сами работники, меняя статус заказа по мере его выполнения. Также частые обращения идут к полю id, по нему работники отсматривают все свои заказы или ищут необходимые, администраторы также следят за конкретными заказами, которые находят по id. Также поле createdAt (дата создания заказа) часто отсматривается администраторами, чтобы следить за новыми или старыми заказами.
В таблице Car часто используемым полем является id, по нему администраторы и работники ищут конкретную машину, над которой производятся манипуляции.
В таблице Offer часто используемым полем является name (название услуги). К нему обращаются клиенты, когда ищут услуги в перечне доступных, а также могут обращаться администраторы и работники для уточнения информации по услуге.
К полям email и phone таблицы Client часто обращаются сотрудники колл-центра для того, чтобы сообщить определенную информацию конкретному клиенту.
Также часто обращения происходят к названию СТО, так как клиент при формировании заказа выбирает место, где получить услугу.

Индексы:
Проведя анализ наиболее часто используемой информации, было принято решение создать следующие индексы:

```
CREATE INDEX IF NOT EXISTS purchase_id ON purchase USING HASH(id);

CREATE INDEX IF NOT EXISTS purchase_status ON purchase USING HASH(state);

CREATE INDEX IF NOT EXISTS purchase_createdat ON purchase USING
BTREE(createdat);
```

К таблице Purchase созданы индексы для работы с id, state, createdAt. Так как для поиска id и статуса заказа всегда используется операция «=», добавлены индексы hash. В случае же поля createdAt был добавлен индекс btree, так как для поиска старых и новых заказов могут использоваться операторы «<» и «>», например при поиске заказа новее определенной даты.

```
CREATE INDEX IF NOT EXISTS offer_name on offer USING HASH(name);
```

К таблице Offer создан индекс hash для поля name. Название услуги всегда ищется по конкретному значение операцией «=».

```
CREATE INDEX IF NOT EXISTS sto_name ON sto USING HASH(name);
```

К таблице Sto создан индекс hash для поля name. Название СТО всегда ищется по конкретному значение операцией «=».

```
CREATE INDEX IF NOT EXISTS detail_name ON detail USING HASH(name);
```

Деталь ищется по конкретному названию, поэтому целесообразно создать индекс для поля имени.

```
CREATE INDEX IF NOT EXISTS tool_name ON tool USING HASH(name);
```

Инструмент ищется по конкретному названию, поэтому целесообразно создать индекс для поля имени.

```
CREATE INDEX IF NOT EXISTS client_id ON client USING HASH(id);
```

В таблице клиент необходимо среди большого количества клиентов находить конкретные номера телефонов и почтовые адреса, поэтому был создан индекс hash для поля id, который ускоряет поиск нужного клиента и информации о нем.

```
CREATE INDEX IF NOT EXISTS car_id ON car USING HASH(id);
```

Для ускорения поиска конкретной машины по ее идентификатору был создан индекс hash для поля id.

Данные индексы покрывают наиболее часто используемые поля, добавление других дополнительных индексов является не целесообразным, потому что перегрузит нашу базу данных при этом не ускорит работу с ней.

Доказательство:
Необходимо учитывать, что в postgres есть одна особенность - когда мы создаем PK столбцы они изначально имеют btree индекс. Поэтому скорость выполнения одного запроса и разница между hash и btree не значительна, но если его выполнять в цикле, то разница достаточно сильно видна (Даже воспользовавшись explain analyze видно, что в место btree индекса, у нас выбирается нами созданный hash индекс так как он эффективнее).

Приведем пример некоторых частых запросов и посмотрим как меняется время выполнения.

Клиент смотрит статус своего заказа.

```
SELECT state FROM purchase
WHERE id = 3;
```

Добавим в нашу таблицу побольше записей, чтобы увидеть работу индекса:

```
studs=> SELECT count(*) FROM purchase;
 count
-------
  1270
(1 строка)
```

До добавления индекса:

```
studs=> EXPLAIN ANALYZE
SELECT state
FROM purchase
WHERE id = 3;
                                                    QUERY PLAN

-------------------------------------------------------------------------------------------------------------------------
-
 Index Scan using purchase_pkey on purchase  (cost=0.28..8.29 rows=1 width=25) (actual time=0.011..0.012 rows=1 loops=1)
   Index Cond: (id = 3)
 Planning Time: 0.196 ms
 Execution Time: 0.036 ms
(4 строки)
```

После добавления индекса:

```
studs=> EXPLAIN ANALYZE
SELECT state
FROM purchase
WHERE id = 3;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------
 Index Scan using purchase_id on purchase  (cost=0.00..8.02 rows=1 width=25) (actual time=0.006..0.006 rows=1 loops=1)
   Index Cond: (id = 3)
 Planning Time: 0.033 ms
 Execution Time: 0.015 ms
(4 строки)
```

Время выполнения запроса ускорилось с 0.036 до 0.016 ms. После добавления индекса был выбран уже наш индекс для поиска, а не btree, который создает сам postgres.

Работник обновляет статус заказа.

```
EXPLAIN ANALYZE UPDATE Purchase
            SET state    = 'Выполнен',
                closedAt = current_date
            WHERE id = 496;
```

До добавления индекса:

```
                                                    QUERY PLAN

-------------------------------------------------------------------------------------------------------------------------
--------
 Update on purchase  (cost=0.28..8.30 rows=0 width=0) (actual time=0.080..0.081 rows=0 loops=1)
   ->  Index Scan using purchase_pkey on purchase  (cost=0.28..8.30 rows=1 width=526) (actual time=0.038..0.040 rows=1 l
oops=1)
         Index Cond: (id = 496)
 Planning Time: 0.122 ms
 Execution Time: 0.117 ms
(5 строк)
```

После добавления индекса:

```
                                                    QUERY PLAN

-------------------------------------------------------------------------------------------------------------------------
------
 Update on purchase  (cost=0.00..8.02 rows=0 width=0) (actual time=0.067..0.067 rows=0 loops=1)
   ->  Index Scan using purchase_id on purchase  (cost=0.00..8.02 rows=1 width=526) (actual time=0.023..0.024 rows=1 loo
ps=1)
         Index Cond: (id = 496)
 Planning Time: 0.338 ms
 Execution Time: 0.098 ms
(5 строк)
```

Время выполнения запроса ускорилось с 0.117 до 0.098 ms. После добавления индекса был выбран уже наш индекс для поиска.

Администратор смотрит заказы, созданные в определенный промежуток времени.

```
SELECT * FROM purchase
WHERE createdat > '2023-05-01' and createdat < '2023-07-01';
```

До добавления индекса:

```
studs=> EXPLAIN ANALYZE SELECT * FROM purchase
WHERE createdat > '2023-05-01' and createdat < '2023-07-01';
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Seq Scan on purchase  (cost=0.00..97.55 rows=397 width=49) (actual time=0.015..0.607 rows=397 loops=1)
   Filter: ((createdat > '2023-05-01'::date) AND (createdat < '2023-07-01'::date))
   Rows Removed by Filter: 3573
 Planning Time: 0.164 ms
 Execution Time: 0.645 ms
(5 строк)
```

После добавления индекса:

```
studs=> EXPLAIN ANALYZE SELECT * FROM purchase
WHERE createdat > '2023-05-01' and createdat < '2023-07-01';

                                                        QUERY PLAN

-------------------------------------------------------------------------------------------------------------------------
-------
 Bitmap Heap Scan on purchase  (cost=8.35..52.30 rows=397 width=49) (actual time=0.065..0.202 rows=397 loops=1)
   Recheck Cond: ((createdat > '2023-05-01'::date) AND (createdat < '2023-07-01'::date))
   Heap Blocks: exact=38
   ->  Bitmap Index Scan on purchase_createdat  (cost=0.00..8.25 rows=397 width=0) (actual time=0.045..0.046 rows=397 lo
ops=1)
         Index Cond: ((createdat > '2023-05-01'::date) AND (createdat < '2023-07-01'::date))
 Planning Time: 0.148 ms
 Execution Time: 0.249 ms
(7 строк)
```

Время выполнения запроса ускорилось с 0.645 до 0.249 ms. После добавления индекса был выбран уже наш индекс для поиска.

Оператор колл-центра ищет клиента, чтобы с ним связаться по номеру телефона.

```
SELECT * FROM client
WHERE id = 5;
```

Добавим в нашу таблицу побольше записей, чтобы увидеть работу индекса:

```
studs=> SELECT count(*) FROM client as number_of_clients;
 count
-------
  3310
(1 строка)
```

До добавления индекса:

```
                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------------
 Index Scan using client_pkey on client  (cost=0.28..8.30 rows=1 width=64) (actual time=0.024..0.026 rows=1 loops=1)
   Index Cond: (id = 5)
 Planning Time: 0.098 ms
 Execution Time: 0.050 ms
(4 строки)
```

После добавления индекса:

```
                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------------
 Index Scan using client_id on client  (cost=0.00..8.02 rows=1 width=64) (actual time=0.005..0.006 rows=1 loops=1)
   Index Cond: (id = 5)
 Planning Time: 0.039 ms
 Execution Time: 0.014 ms
(4 строки)
```

Время выполнения запроса ускорилось с 0.050 до 0.014 ms. После добавления индекса был выбран уже наш индекс для поиска.

Вывод:

Во время выполнения курсовой работы были изучены работа функций, процедур и триггеров, которые их вызывают, для реализации ограничения целостности. Использованы индексы для ускорения обработки запросов в будущем приложении и проанализирована база данных.