**Отчет**

Курсовая работа №3

**Выполнила:**

Ван Минчжи

**Группа:**

Р33151

**Преподаватель:**

Байрамова Хумай Бахруз Кызы

**Этап 3**

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.**Бизнес-процесс**

• Выбор и заказ продукции:

Покупатели (пользователи приложения) могут использовать app нашей компании.

• Управление заказами:

Минимальное отслеживание статуса заказа

• Информация об изменении статуса заказа.

• Оплата:

Если заказ размещен, покупатель (пользователь приложения) платит немедленно.

# **DDL：**

-- create table

create table if not exists "empolyee"  
(  
 "id" serial primary key,  
 "name" varchar(32) not null  
);  
  
create table if not exists "client"  
(  
 "id" serial primary key,  
 "name" varchar(32) not null  
);  
  
create table if not exists "login0"  
(  
 "id" serial primary key,  
 "code" text not null,  
 "client\_id" integer references client(id)  
);  
  
  
create table if not exists "supplier"  
(  
 "id" serial primary key,  
 "name" varchar(32) not null  
);  
  
create table if not exists "wholesaler"  
(  
 "id" serial primary key,  
 "name" varchar(32) not null  
);  
  
create table if not exists "online\_store"  
(  
 "id" serial primary key,  
 "adress" text not null  
);  
  
create table if not exists "offline\_store"  
(  
 "id" serial primary key,  
 "adress" text not null  
);  
  
create table if not exists "self\_sale"  
(  
 "id" integer primary key,  
 "online\_store\_id" integer references online\_store(id),  
 "offline\_store\_id" integer references offline\_store(id)  
);  
  
create table if not exists "storehouse"  
(  
 "id" serial primary key,  
 "supplier\_id" integer references supplier(id),  
 "wholesaler\_id" integer references wholesaler(id),  
 "self\_sale\_id" integer references self\_sale(id),  
 UNIQUE("id","supplier\_id","wholesaler\_id","self\_sale\_id")  
);  
  
create table if not exists "order"  
(  
"id" serial primary key,  
 "empolyee\_id" integer references empolyee(id),  
 "client\_id" integer references client(id),  
"storehouse\_id" integer references storehouse(id),  
"status" text,  
UNIQUE ("id","empolyee\_id", "client\_id","storehouse\_id","status")  
);

-- drop table

drop table "order" cascade ;  
drop table "storehouse" cascade;  
drop table "self\_sale" cascade;  
drop table "offline\_store" cascade;  
drop table "online\_store" cascade;  
drop table "wholesaler" cascade;  
drop table "supplier" cascade;  
drop table "login0" cascade;  
drop table "client" cascade;  
drop table "empolyee" cascade;

**-- drop index:**

drop index if exists IND\_ORDER\_EMPLOYEE;  
drop index if exists IND\_LOGIN0\_CLIENT;  
drop index if exists IND\_STORE\_SUP;  
drop index if exists IND\_STORE\_WHOLE;  
drop index if exists IND\_SELF\_ONLINE;  
drop index if exists IND\_SELF\_OFFLINE;

# **Заполнение таблиц:**

create or replace function fill\_empolyee() returns void as $$  
DECLARE  
 i int = 0;  
 names varchar(32)[] = array ['Blazer', 'Haven', 'Tony','Peter','Bob','John','Tomas','Alex','Anna'];  
BEGIN  
 while i < 1000000 loop  
 insert into "empolyee"(name)  
 values (names[i%9+1]);  
 i = i + 1;  
 end loop;AKA  
end;  
$$ language plpgsql;  
select *fill\_empolyee*();  
  
  
create or replace function fill\_client() returns void as $$  
DECLARE  
 i int = 0;  
 names varchar(32)[] = array ['Aim', 'Bim', 'Cim','Dim','Eim','Fim','Gim','Him','Iim'];  
BEGIN  
 while i < 1000000 loop  
 insert into "client"(name)  
 values (names[i%9+1]);  
 i = i + 1;  
 end loop;  
end;  
$$ language plpgsql;  
select *fill\_client*();  
  
  
create or replace function fill\_login0() returns void as $$  
DECLARE  
 i int = 0;  
BEGIN  
 while i < 1000000 loop  
 insert into "login0"(code, client\_id)  
 values ('11223344', i%1000000 + 1);  
 i = i + 1;  
 end loop;  
end;  
$$ language plpgsql;  
select *fill\_login0*();  
  
  
create or replace function fill\_supplier() returns void as $$  
DECLARE  
 i int = 0;  
 names varchar(32)[] = array ['Nike', 'Nike', 'Adidas','Reebok','X','361c'];  
BEGIN  
 while i < 1000000 loop  
 insert into "supplier"(name)  
 values (names[i%6+1]);  
 i = i + 1;  
 end loop;  
end;  
$$ language plpgsql;  
select *fill\_supplier*();  
  
  
create or replace function fill\_wholesaler() returns void as $$  
DECLARE  
 i int = 0;  
 names varchar(32)[] = array ['ShiHuo', 'poison', 'AKA','BKB','CKC','DKD'];  
BEGIN  
 while i < 1000000 loop  
 insert into "wholesaler"(name)  
 values (names[i%6+1]);  
 i = i + 1;  
 end loop;  
end;  
$$ language plpgsql;  
select *fill\_wholesaler*();  
  
  
create or replace function fill\_online\_store() returns void as $$  
DECLARE  
 i int = 0;  
 adresses varchar(32)[] = array ['www.store1.com', 'www.store2.com', 'www.store3.com','www.store4.com','www.store5.com','www.store6.com'];  
BEGIN  
 while i < 1000000 loop  
 insert into "online\_store"(adress)  
 values (adresses[i%6+1]);  
 i = i + 1;  
 end loop;  
end;  
$$ language plpgsql;  
select *fill\_online\_store*();  
  
  
create or replace function fill\_offline\_store() returns void as $$  
DECLARE  
 i int = 0;  
 adresses varchar(32)[] = array ['Nevsky Prospekt', 'Romanosov street', 'Albat street','Krovel street', 'AAA street','BBB street','CCC street'];  
BEGIN  
 while i < 1000000 loop  
 insert into "offline\_store"(adress)  
 values (adresses[i%7+1]);  
 i = i + 1;  
 end loop;  
end;  
$$ language plpgsql;  
select *fill\_offline\_store*();  
  
  
create or replace function fill\_self\_sale() returns void as $$  
DECLARE  
 i int = 0;  
BEGIN  
 while i < 1000000 loop  
 insert into "self\_sale"(online\_store\_id, offline\_store\_id)  
 values (i%1000000 + 1, i%1000000 + 1);  
 i = i + 1;  
 end loop;  
end;  
$$ language plpgsql;  
select *fill\_self\_sale*();  
  
  
create or replace function fill\_storehouse() returns void as $$  
DECLARE  
 i int = 0;  
BEGIN  
 while i < 1000000 loop  
 insert into "storehouse"(supplier\_id,wholesaler\_id,self\_sale\_id)  
 values (i%1000000 + 1, i%1000000 + 1, i%1000000 + 1);  
 i = i + 1;  
 end loop;  
end;  
$$ language plpgsql;  
select *fill\_storehouse*();  
  
  
create or replace function fill\_order() returns void as $$  
DECLARE  
 i int = 0;  
BEGIN  
 while i < 1000000 loop  
 insert into "order"(empolyee\_id,client\_id,storehouse\_id,status)  
 values (i%1000000 + 1, i%1000000 + 1,i%1000000 + 1, 'Regular');  
 i = i + 1;  
 end loop;  
end;  
$$ language plpgsql;  
select *fill\_order*();

# **Functions:**

CREATE OR REPLACE FUNCTION generate\_unique\_id() RETURNS INTEGER AS $$  
DECLARE  
 new\_id INTEGER;  
BEGIN  
 SELECT *MAX*(id) + 1 INTO new\_id FROM "login0";  
 IF new\_id IS NULL THEN  
 new\_id := 1;  
 END IF;  
 RETURN new\_id;  
END;  
$$ LANGUAGE plpgsql;  
-- 用户注册  
CREATE OR REPLACE FUNCTION register(name\_u VARCHAR(64), password\_u VARCHAR(256))   
RETURNS SETOF "client" AS $$  
DECLARE  
 new\_id INT;  
BEGIN  
 new\_id := generate\_unique\_id();  
 INSERT INTO "client" (id, name) VALUES (new\_id, name\_u);  
 INSERT INTO "login0" (id, client\_id, code) VALUES (new\_id, new\_id, password\_u);  
 RETURN QUERY SELECT \* FROM "client" WHERE id = new\_id;  
END;  
$$ LANGUAGE plpgsql;  
  
-- 查看所有订单状态  
CREATE OR REPLACE FUNCTION view\_order\_status()   
RETURNS TABLE (status TEXT) AS $$  
BEGIN  
 RETURN QUERY   
 SELECT "order".status FROM "order";  
END;  
$$ LANGUAGE plpgsql;  
  
-- 订单支付  
CREATE OR REPLACE FUNCTION generate\_unique\_id\_order() RETURNS INTEGER AS $$  
DECLARE  
 new\_id INTEGER;  
BEGIN  
 SELECT *MAX*(id) + 1 INTO new\_id FROM "order";  
 IF new\_id IS NULL THEN  
 new\_id := 1;  
 END IF;  
 RETURN new\_id;  
END;  
$$ LANGUAGE plpgsql;  
  
  
CREATE OR REPLACE FUNCTION pay(empolyee\_id integer,client\_id integer,storehouse\_id integer,status text)  
RETURNS SETOF "order" AS $$  
DECLARE  
 new\_id INT;  
BEGIN  
 new\_id := generate\_unique\_id\_order();  
 INSERT INTO "order" (id,empolyee\_id,client\_id,storehouse\_id,status) VALUES (new\_id, empolyee\_id,client\_id,storehouse\_id,status);  
 RETURN QUERY SELECT \* FROM "order" WHERE id = new\_id;  
END;  
$$ LANGUAGE plpgsql;

# **Trigger:**

CREATE OR REPLACE FUNCTION check\_order\_status()  
RETURNS TRIGGER AS $$  
BEGIN  
 IF NEW.id > 3 THEN  
 NEW.status = 'High Value';  
 ELSE  
 NEW.status = 'Regular';  
 END IF;  
 RETURN NEW;  
END;  
$$ LANGUAGE plpgsql;  
  
CREATE TRIGGER enforce\_order\_status  
BEFORE INSERT ON "order"  
FOR EACH ROW  
EXECUTE FUNCTION check\_order\_status();

# **Индекс:**

CREATE INDEX IND\_ORDER\_EMPLOYEE ON "order"(empolyee\_id);

CREATE INDEX IND\_LOGIN0\_CLIENT ON LOGIN0(CLIENT\_ID);  
CREATE INDEX IND\_SELF\_ONLINE ON SELF\_SALE(ONLINE\_STORE\_ID);  
CREATE INDEX IND\_SELF\_OFFLINE ON SELF\_SALE(OFFLINE\_STORE\_ID);  
CREATE INDEX IND\_STORE\_SUP ON STOREHOUSE(SUPPLIER\_ID);  
CREATE INDEX IND\_STORE\_WHOLE ON STOREHOUSE(SWHOLESALER\_ID);

# **Наиболее часто используемыезапросык объектам базы данных:**

1. insert into Empolyee  
   insert into client  
   insert into login0  
   insert into supplier  
   insert into wholesaler  
   insert into online\_store  
   insert into offline\_store  
   insert into self\_sale  
   insert into storehouse  
   insert into "order"

Скрипт – это главная функция нашего приложения. Пользователи или программисты создают скриптипользователи их используют. Высшие запросывседля создания скриптов

1. insert into "order"  
   SELECT \* FROM "order";

Этот код используется для запроса статуса заказа добавленных заказов и всех заказов.

1. INSERT INTO "client" (id, name) VALUES (new\_id, name\_u);  
   INSERT INTO "login0" (id, client\_id, code) VALUES (new\_id, new\_id, password\_u);  
   SELECT \* FROM "client";

Этот код используется для регистрации и добавления пользователей, установки их имен и паролей, а также запроса всех пользователей.

1. CREATE OR REPLACE FUNCTION generate\_unique\_id\_order() RETURNS INTEGER AS $$  
   DECLARE  
    new\_id INTEGER;  
   BEGIN  
    SELECT *MAX*(id) + 1 INTO new\_id FROM "order";  
    IF new\_id IS NULL THEN  
    new\_id := 1;  
    END IF;  
    RETURN new\_id;  
   END;  
   $$ LANGUAGE plpgsql;

Этот код используется для генерации уникального значения идентификатора заказа. Каждый раз, когда эта функция вызывается, будет возвращено новое уникальное значение идентификатора заказа, которое можно использовать для вставки новых записей заказов, избегая проблемы конфликтов первичных ключей.

1. CREATE OR REPLACE FUNCTION generate\_unique\_id() RETURNS INTEGER AS $$  
   DECLARE  
    new\_id INTEGER;  
   BEGIN  
    SELECT *MAX*(id) + 1 INTO new\_id FROM "login0";  
    IF new\_id IS NULL THEN  
    new\_id := 1;  
    END IF;  
    RETURN new\_id;  
   END;  
   $$ LANGUAGE plpgsql;

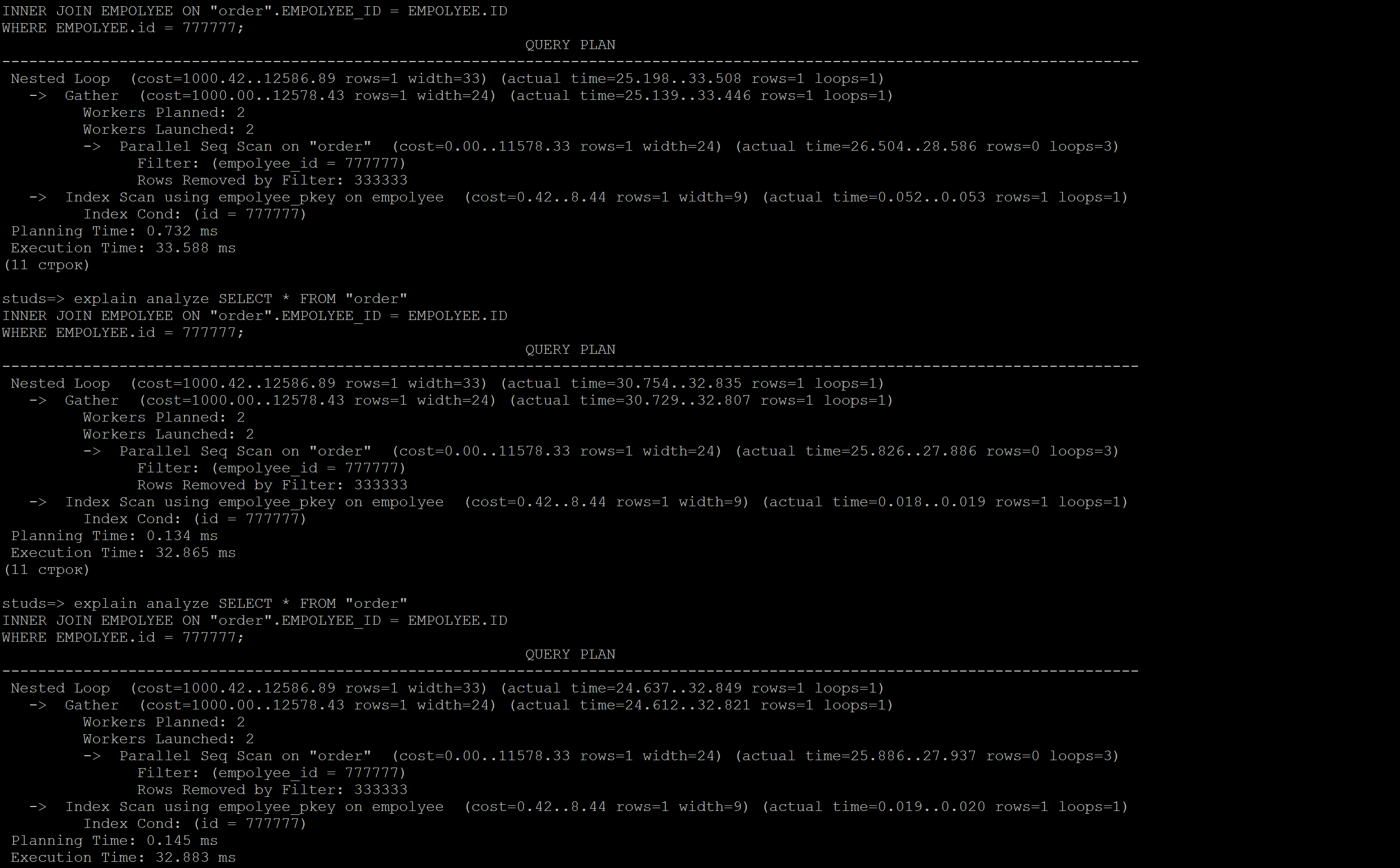
Это тоже самое с 4.

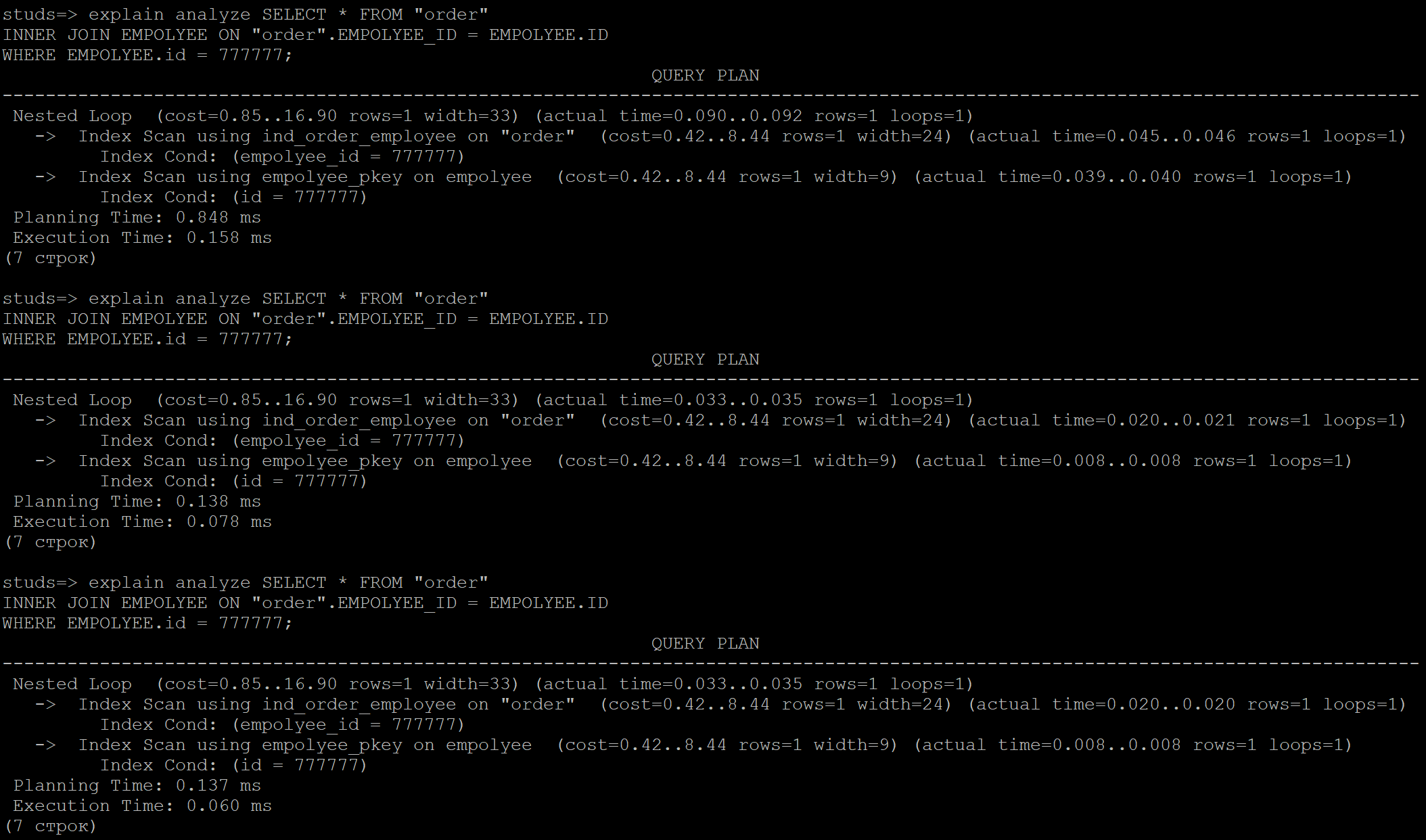
# **Доказание полезности индесков.**

1. IND\_ORDER\_EMPOLYEE

SELECT \* FROM "order"  
INNER JOIN EMPOLYEE ON "order".EMPOLYEE\_ID = EMPOLYEE.ID  
WHERE EMPOLYEE.ID = 777777;

|  |  |  |  |
| --- | --- | --- | --- |
| index | 1-ый | 2-ой | 3-ий |
| c | 0.158 | 0.078 | 0.060 |
| без индекса | 33.588 | 32.865 | 32.883 |

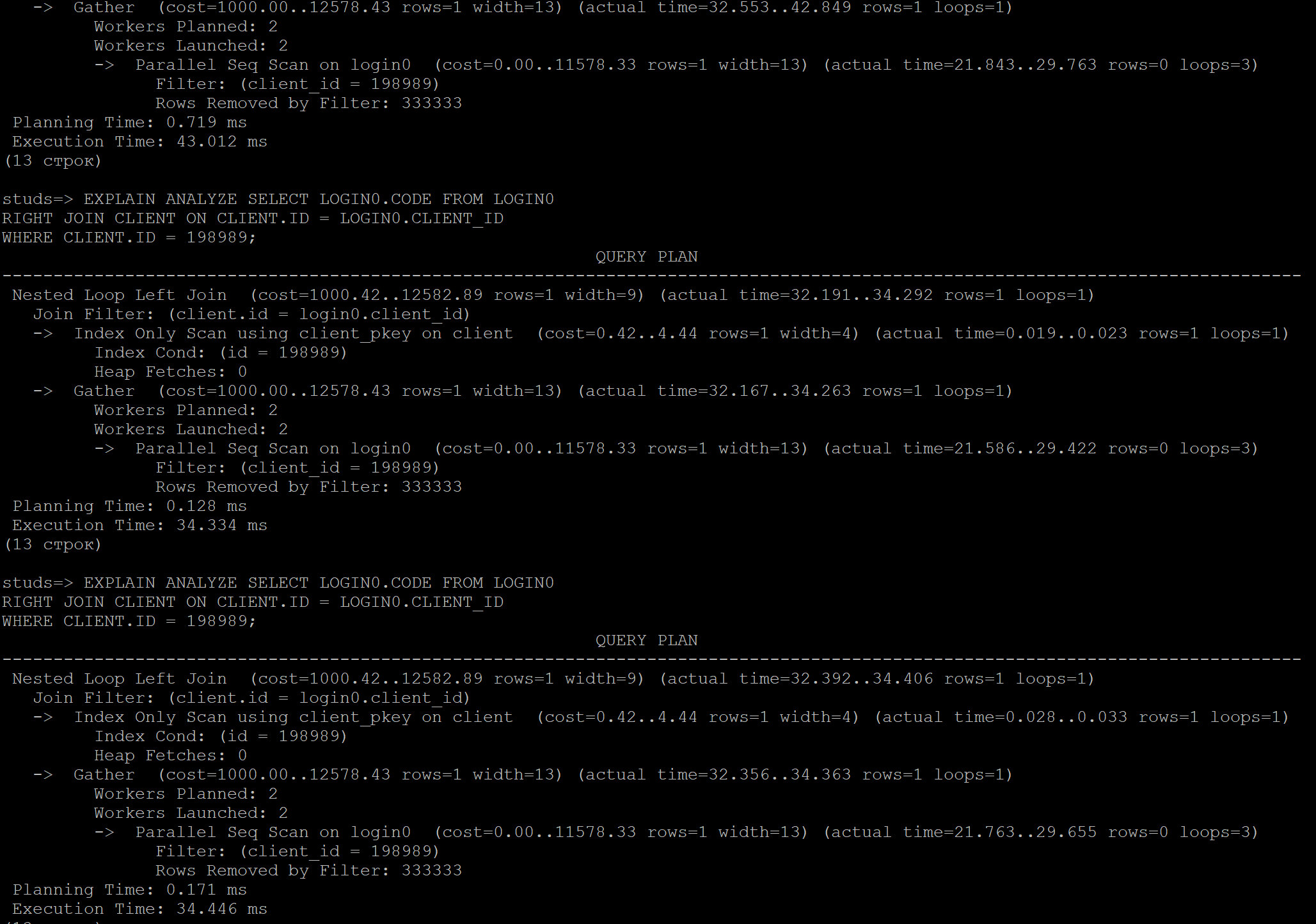


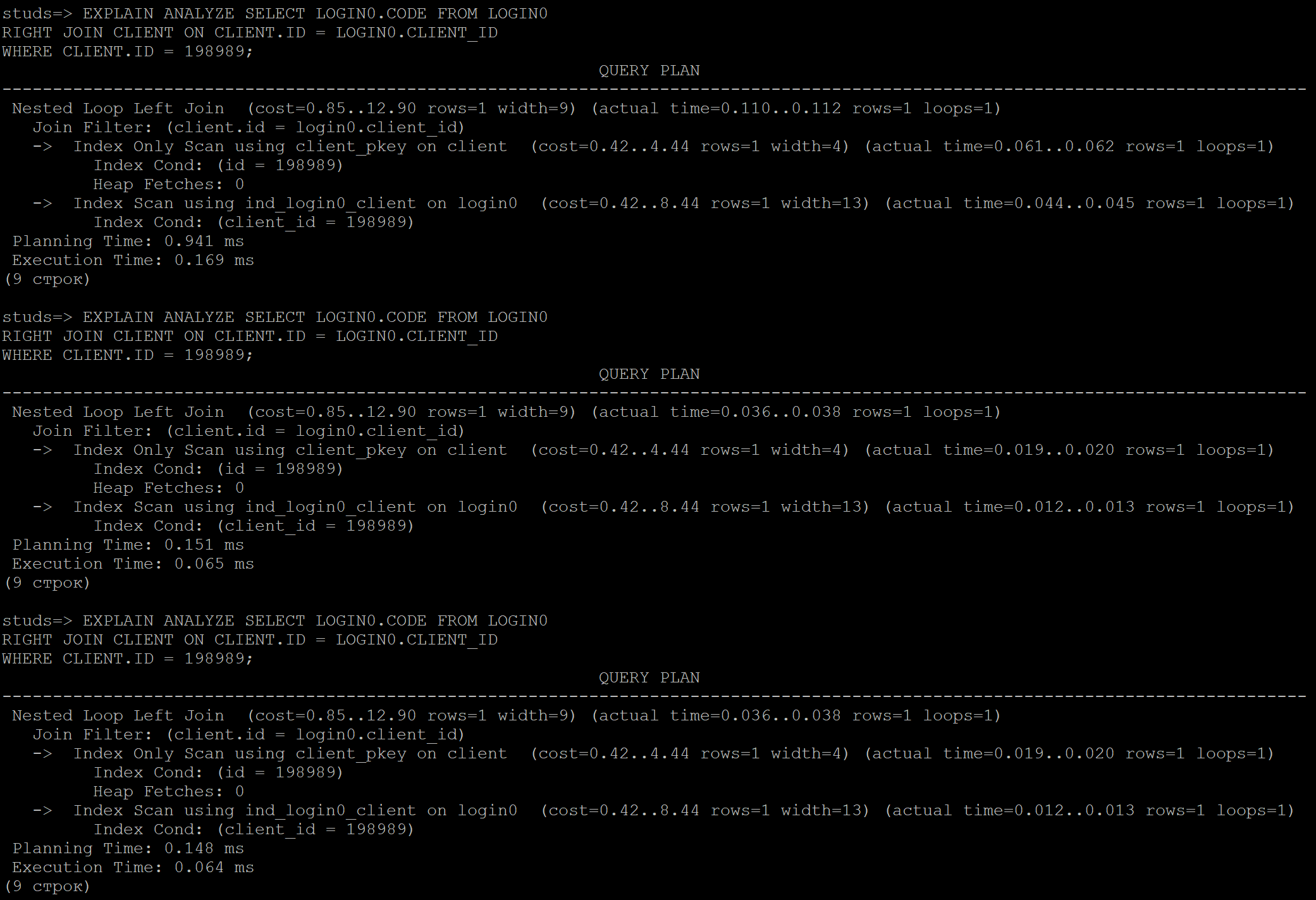


1. IND\_LOGIN0\_CLIENT

SELECT LOGIN0.CODE FROM LOGIN0  
RIGHT JOIN CLIENT ON CLIENT.ID = LOGIN0.CLIENT\_ID  
WHERE CLIEND.ID = 198989;

|  |  |  |  |
| --- | --- | --- | --- |
| index | 1-ый | 2-ой | 3-ий |
| без индекса | 0.169 | 0.065 | 0.064 |
| с индексом | 43.012 | 34.334 | 34.446 |

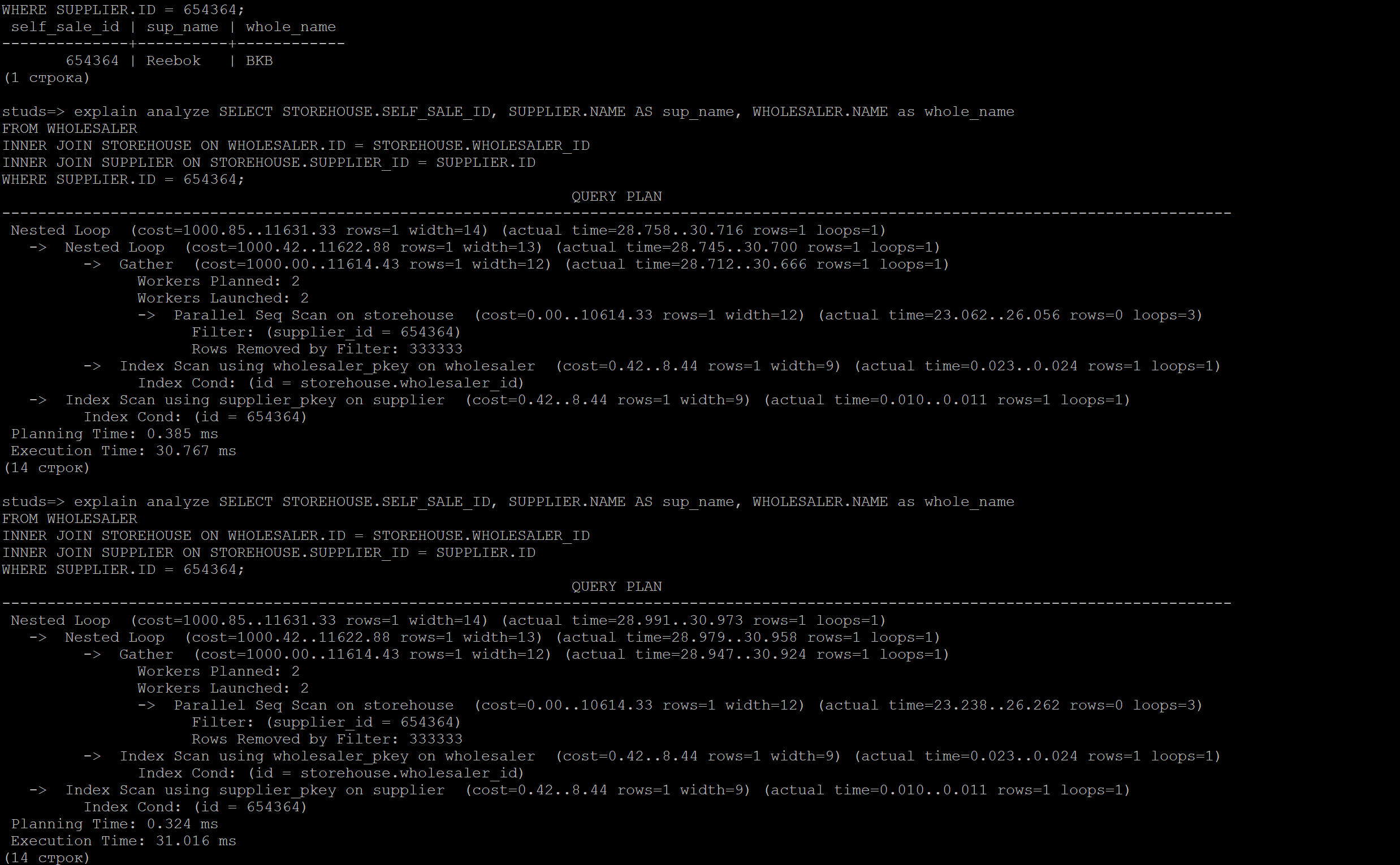


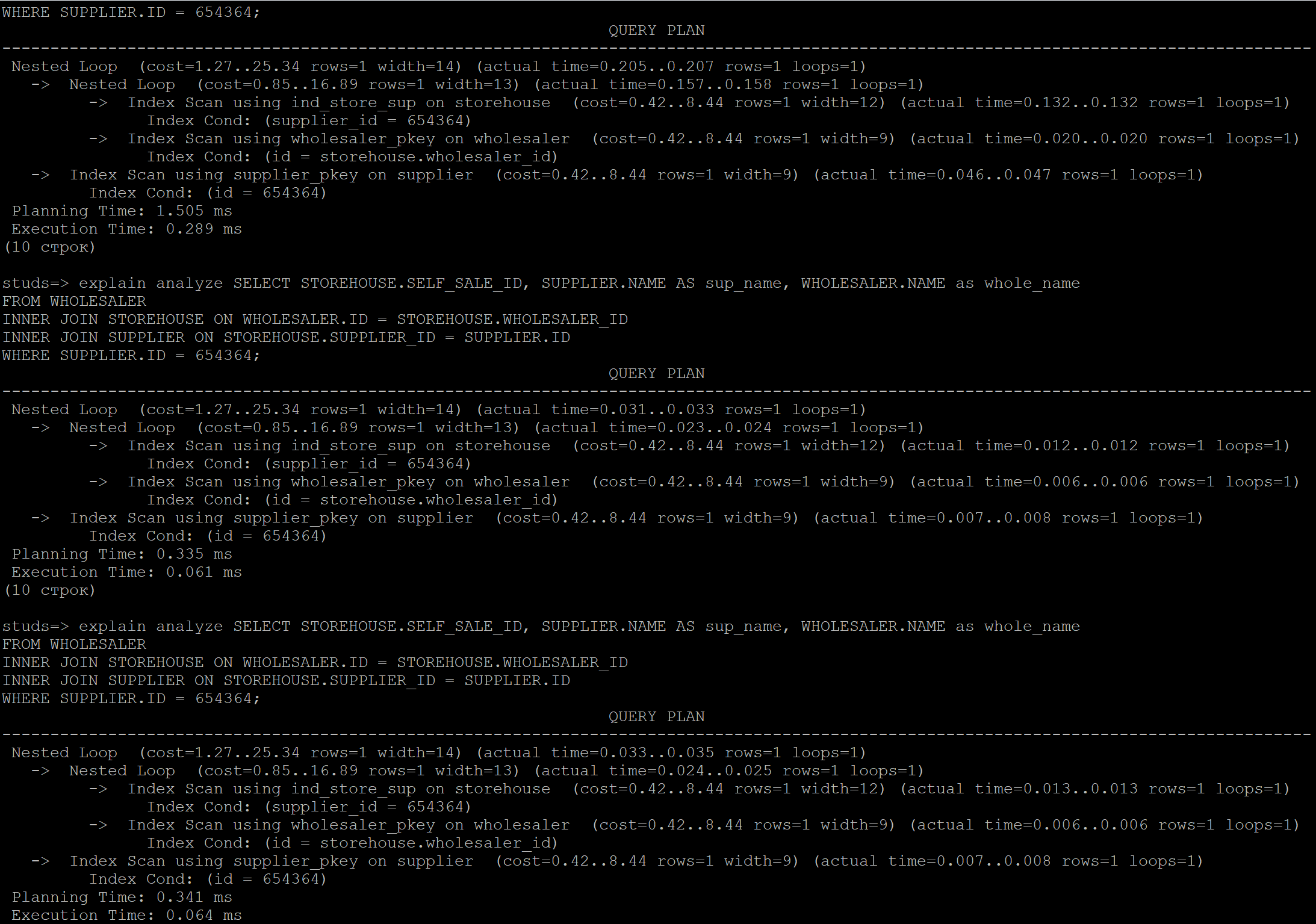


1. IND\_STORE\_SUP & IND\_STORE\_WHOLE

SELECT STOREHOUSE.SELF\_SALE\_ID, SUPPLIER.NAME AS sup\_name, WHOLESALER.NAME as whole\_name  
FROM WHOLESALER  
INNER JOIN STOREHOUSE ON WHOLESALER.ID = STOREHOUSE.WHOLESALER\_ID  
INNER JOIN SUPPLIER ON STOREHOUSE.SUPPLIER\_ID = SUPPLIER.ID  
WHERE SUPPLIER.ID = 654364;

|  |  |  |  |
| --- | --- | --- | --- |
| index | 1-ый | 2-ой | 3-ий |
| без индекса | 0.289 | 0.061 | 0.064 |
| с индексом | 30.767 | 31.016 | 31.039 |





1. IND\_SELF\_ONLINE & IND\_SELF\_OFFLINE

SELECT SELF\_SALE.ID AS SS\_ID, OFFLINE\_STORE.ADRESS AS OFF\_ADRESS  
FROM ONLINE\_STORE  
INNER JOIN SELF\_SALE ON ONLINE\_STORE.ID = SELF\_SALE.ONLINE\_STORE\_ID  
INNER JOIN OFFLINE\_STORE ON SELF\_SALE.OFFLINE\_STORE\_ID = OFFLINE\_STORE.ID  
WHERE ONLINE \_STORE.ID = 123452;

|  |  |  |  |
| --- | --- | --- | --- |
| index | 1-ый | 2-ой | 3-ий |
| без индекса | 0.338 | 0.065 | 0.067 |
| с индексом | 30.322 | 30.451 | 30.329 |

