Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовая работа по курсу**

**«Базы данных»**

**Тема: Каталог ноутбуков**

Выполнили: Лисин Р.С.,

Мальцев И.Д., Фролов М.А.

Группа: М8О-306Б-20

Преподаватель: Сеченых П.А.

Дата: \_\_\_\_\_\_\_\_\_\_\_

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Таблицы**

Таблица laptops:

* laptop\_id – идентификатор ноутбука (SERIAL PRIMARY KEY)
* laptop\_name – имя ноутбука (VARCHAR(50) NOT NULL)
* producer – производитель ноутбука (VARCHAR(25) NOT NULL)
* price – цена ноутбука (NUMERIC NOT NULL)
* characteric – характеристики ноутбука (TEXT NULL)

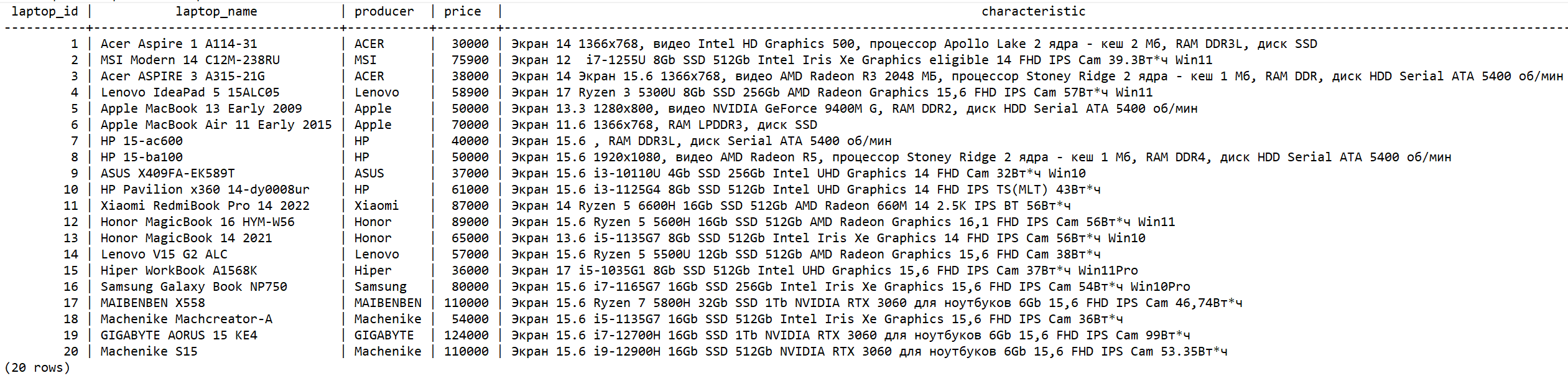


Таблица providers:

* provider\_id – идентификатор поставщика (SERIAL PRIMARY KEY)
* provide\_name – название поставщика (VARCHAR(25) NOT NULL)
* address – адрес поставщика (VARCHAR(50) NOT NULL)
* phone – телефон поставщика (VARCHAR(20) NULL)
* reputation – репутация поставщика (NUMERIC NULL)

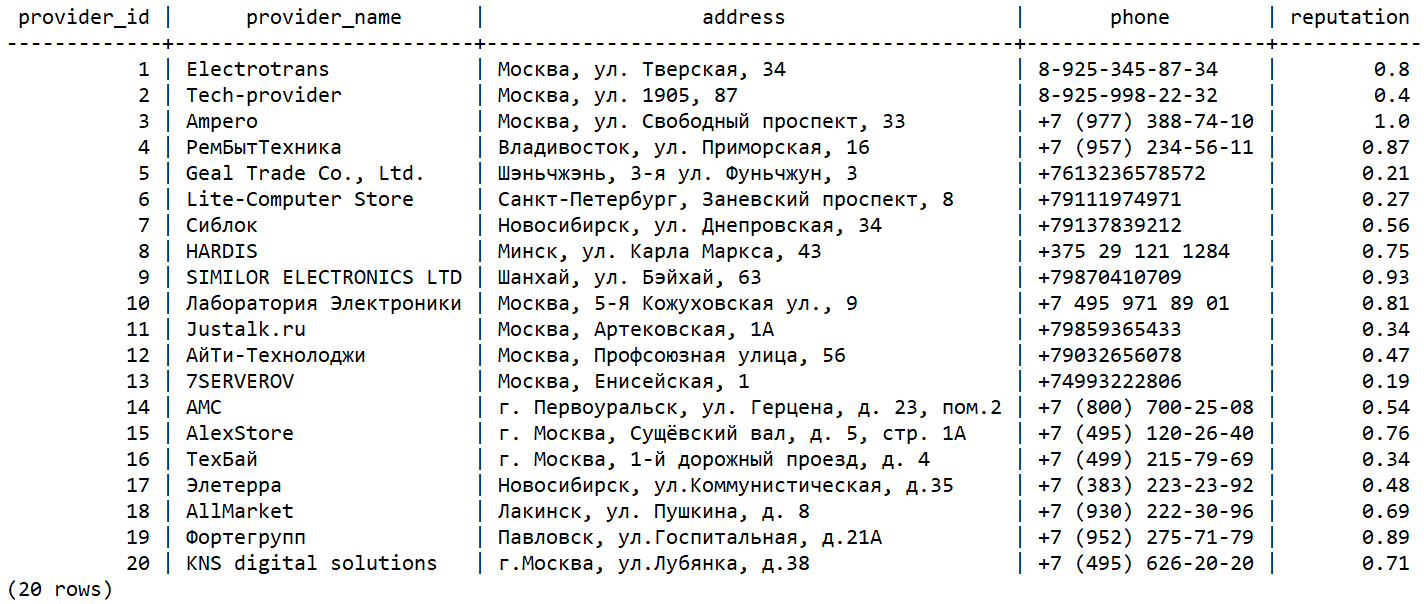


Таблица shops:

* shop\_id – идентификатор магазина (INTEGER NOT NULL)
* shop\_name – идентификатор магазина (VARCHAR(25) NOT NULL)
* last\_delivery – дата последней доставки в магазин данного товара (TIMESTAMPTZ NULL)
* shop\_address – адрес магазина (VARCHAR(50) NOT NULL)
* quantity – количество товара данного типа (INTEGER NULL)
* laptop\_id – идентификатор ноутбука (INTEGER NOT NULL REFERENCES laptops (laptop\_id))
* provider\_id – идентификатор поставщика (INTEGER NOT NULL REFERENCES providers (provider\_id))

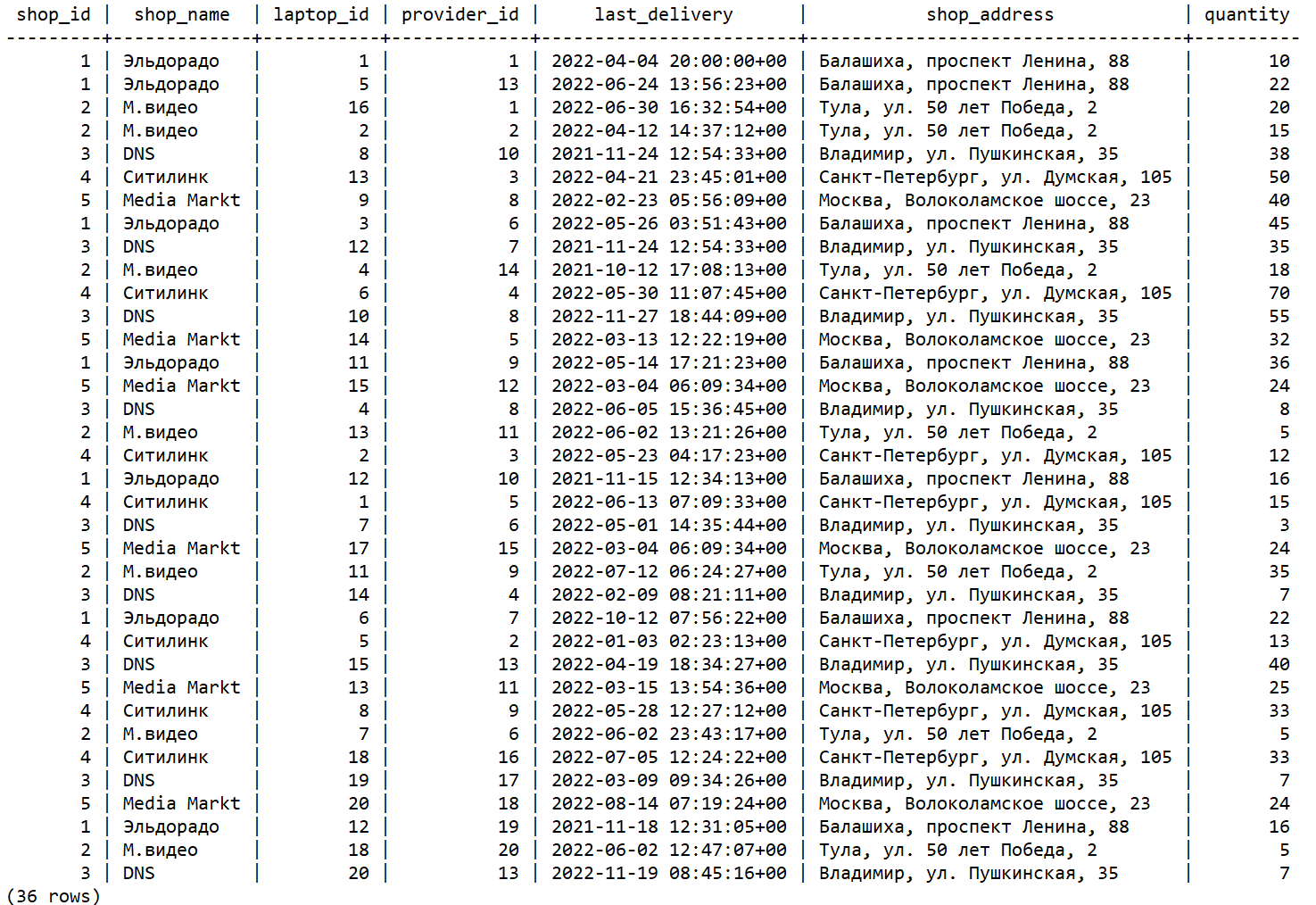


Таблица laptops\_audits:

* laptop\_name – имя ноутбука (VARCHAR(50) NOT NULL)
* producer – производитель ноутбука (VARCHAR(25) NOT NULL)
* price – предыдущая цена ноутбука (NUMERIC NOT NULL)
* characteric – характеристики ноутбука (TEXT NULL)

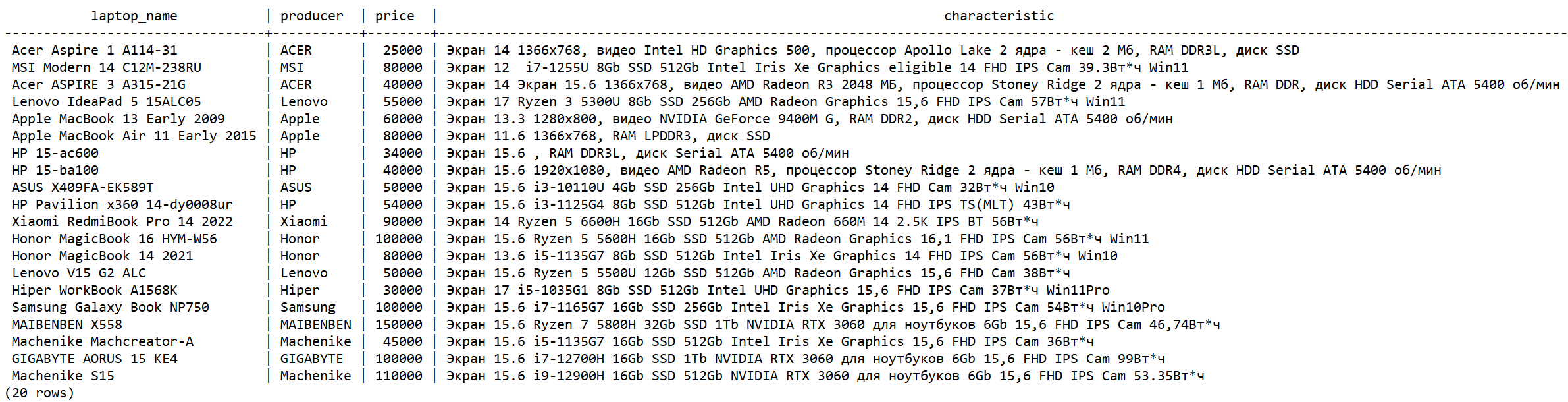


Таблица customers:

* customer\_id - идентификатор покупателя (SERIAL PRIMARY KEY)
* customer\_name – имя покупателя (VARCHAR(50) NOT NULL)
* shop\_id – идентификатор магазина, в котором совершена покупка (INTEGER NOT NULL)
* laptop\_id – идентификатор купленного ноутбука (INTEGER NOT NULL REFERENCES laptops (laptop\_id))
* customer\_balance - денежный баланс покупателя (NUMERIC NOT NULL)



Таблица sellers:

* seller\_id - идентификатор покупателя (SERIAL PRIMARY KEY)
* seller\_name – имя покупателя (VARCHAR(50) NOT NULL)
* shop\_id – идентификатор магазина, в котором совершена покупка (INTEGER NOT NULL)
* laptop\_id – идентификатор купленного ноутбука (INTEGER NOT NULL REFERENCES laptops (laptop\_id))
* seller\_balance - денежный баланс продавца (NUMERIC NOT NULL)



Первичные ключи:

* для таблицы laptops – laptops\_id
* для таблицы providers – provider\_id
* для таблицы shops – shop\_id
* для таблицы customers – customer\_id
* для таблицы sellers – seller\_id

Ключи связи присутствуют только в таблице shops:

* laptop\_id из таблицы laptops
* provider\_id из таблицы providers

*Код для генерации БД:*

CREATE TABLE laptops (

laptop\_id SERIAL PRIMARY KEY,

laptop\_name VARCHAR(50) NOT NULL,

producer VARCHAR(25) NOT NULL,

price NUMERIC NOT NULL,

characteristic TEXT NULL

);

CREATE TABLE providers (

provider\_id SERIAL PRIMARY KEY,

provider\_name VARCHAR(25) NOT NULL,

address VARCHAR(50) NOT NULL,

phone VARCHAR(20) NULL,

reputation NUMERIC NULL

);

CREATE TABLE shops (

shop\_id INTEGER NOT NULL,

shop\_name VARCHAR(25) NOT NULL,

laptop\_id INTEGER NOT NULL REFERENCES laptops (laptop\_id),

provider\_id INTEGER NOT NULL REFERENCES providers (provider\_id),

last\_delivery TIMESTAMPTZ NULL,

shop\_address VARCHAR(40) NOT NULL,

quantity INTEGER NULL

);

CREATE UNIQUE INDEX laptop\_idx ON laptops(laptop\_id);

CREATE UNIQUE INDEX provider\_idx ON providers(provider\_id);

CREATE INDEX laptop\_provider\_idx ON shops(laptop\_id, provider\_id);

CREATE TABLE laptops\_audits (

laptop\_name VARCHAR(50) NOT NULL,

producer VARCHAR(25) NOT NULL,

price NUMERIC NOT NULL,

characteristic TEXT NULL

);

CREATE TABLE customers (

customer\_id SERIAL PRIMARY KEY,

customer\_name VARCHAR(50) NOT NULL,

shop\_id INTEGER NOT NULL,

laptop\_id INTEGER NOT NULL REFERENCES laptops (laptop\_id),

customer\_balance NUMERIC NOT NULL

);

CREATE TABLE sellers (

seller\_id SERIAL PRIMARY KEY,

seller\_name VARCHAR(50) NOT NULL,

shop\_id INTEGER NOT NULL,

laptop\_id INTEGER NOT NULL REFERENCES laptops (laptop\_id),

seller\_balance NUMERIC NOT NULL

);

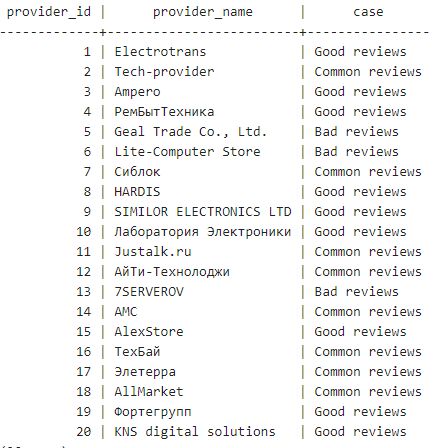
CREATE UNIQUE INDEX customers\_idx ON customers(customer\_id);

CREATE UNIQUE INDEX sellers\_idx ON sellers(seller\_id);

**Запросы к базе данных**

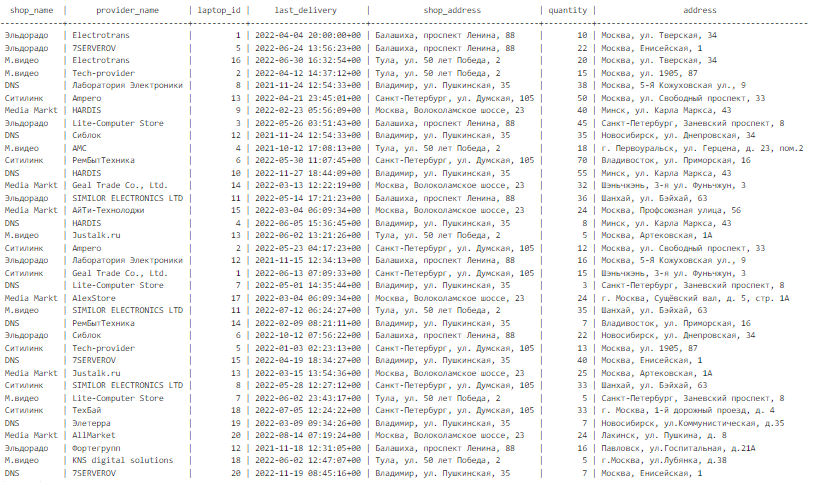
1. Получение уровня рейтинга поставщика.

SELECT provider\_id, provider\_name, CASE WHEN reputation=NULL THEN 'No reviews' WHEN reputation < 0.3 THEN 'Bad reviews' WHEN reputation < 0.7 THEN 'Common reviews' ELSE 'Good reviews' END from providers;



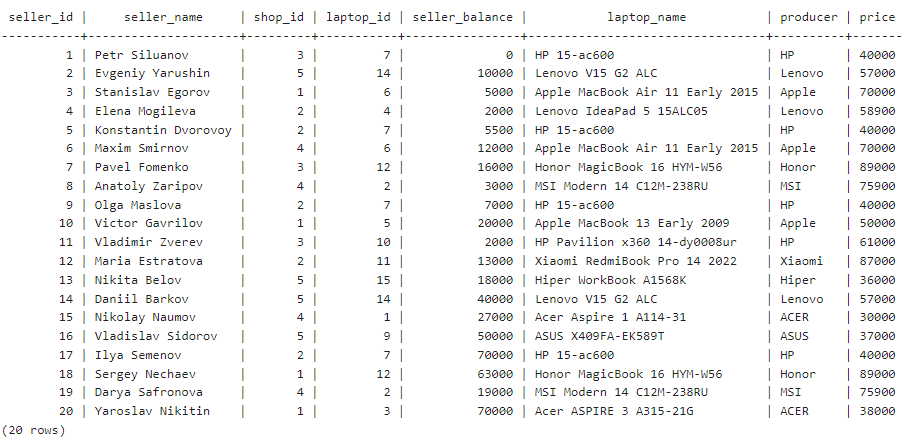
2. Выдача названия магазина, имени поставщика, идентификатора ноутбука и даты последней доставки при объединении таблиц shops и providers.

select shop\_name,provider\_name, laptop\_id, last\_delivery, shop\_address, quantity, address from shops join providers on shops.provider\_id=providers.provider\_id;



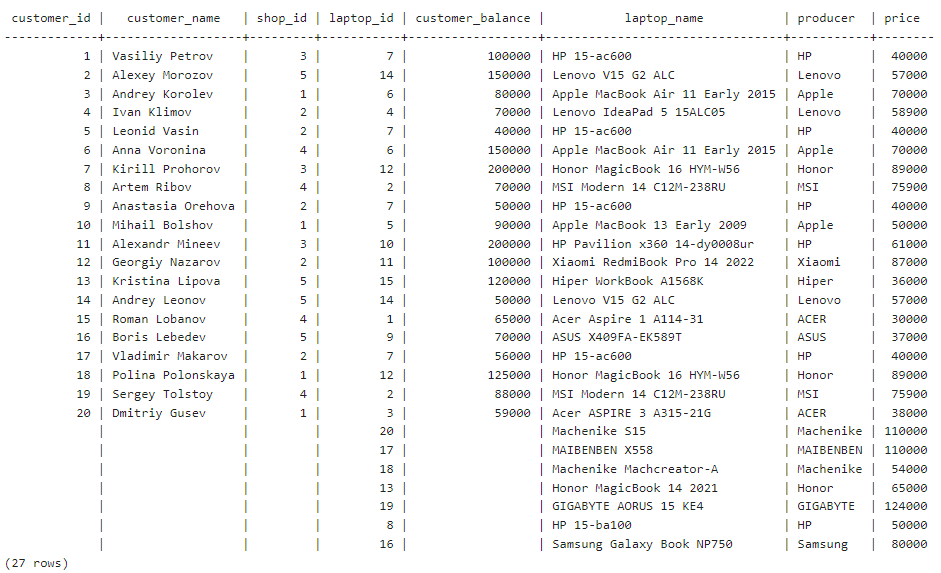
3. Для каждой записи продавца определение проданного ноутбука.

select seller\_id, seller\_name, shop\_id, laptops.laptop\_id, seller\_balance, laptop\_name,producer, price from sellers left join laptops on sellers.laptop\_id=laptops.laptop\_id;



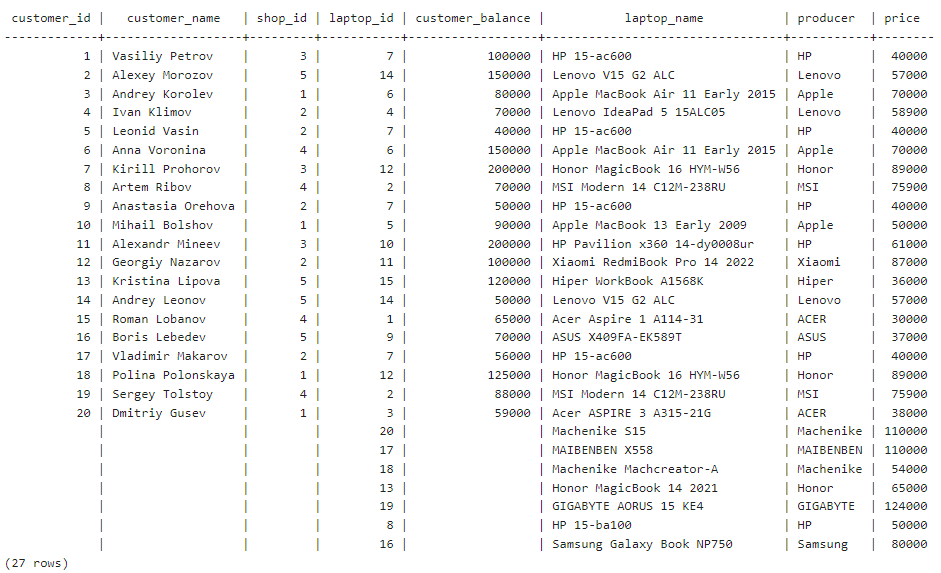
4. Для каждого ноутбука определить продавца, продавшего его (если есть).

select customer\_id, customer\_name, shop\_id, laptops.laptop\_id, customer\_balance, laptop\_name,producer, price from customers right join laptops on customers.laptop\_id=laptops.laptop\_id;



4.Выдача для каждого для каждого пользователя моделей ноутбуков, которые он не сможет себе позволить и цена которых большее 100`000.

select customer\_name,customer\_balance,laptop\_name,price from customers cross join laptops where customers.customer\_balance < laptops.price AND laptops.price > 100000;

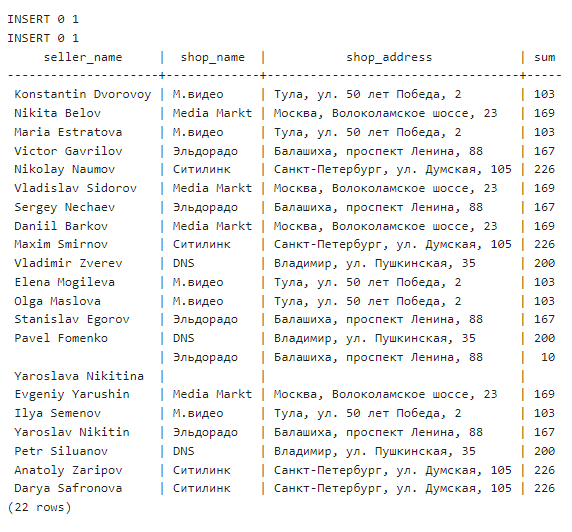


5.Получение соответствия, сколько ноутбуков какой продавец продал в каком магазине.

INSERT INTO sellers values (21, 'Yaroslava Nikitina', 31, 3, 70000);

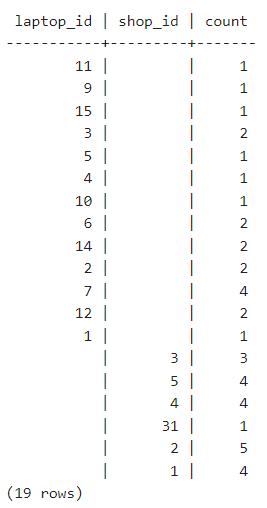
INSERT INTO shops VALUES (33, 'Эльдорадо', 1, 1, '2022-04-04 20:00:00', 'Балашиха, проспект Ленина, 88', 10);

select seller\_name, shop\_name, shop\_address, sum(quantity) from sellers full join shops on sellers.shop\_id=shops.shop\_id group by seller\_name, shop\_name, shop\_address;



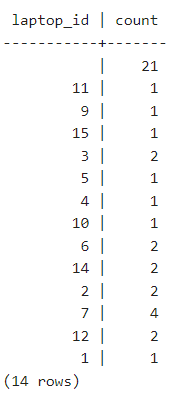
6.Вычисление количества использования магазинов и ноутбуков, использованных в таблице sellers.

select laptop\_id,shop\_id, count(\*) from sellers group by grouping sets(laptop\_id,shop\_id);



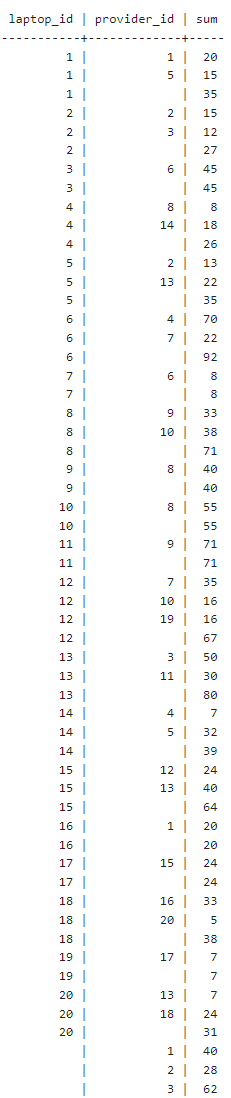
7.Вычисление общего количества ноутбуков, которые использовались в таблице sellers и для каждого подсчитать, сколько раз он использовался.

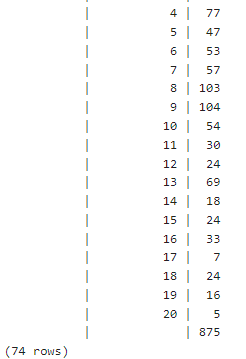
select laptop\_id, count(\*) from sellers group by rollup(laptop\_id);



8. Выдача общего количества ноутбуков и в разбивке по идентификатор ноутбука – поставщик.

select laptop\_id, provider\_id, sum(quantity) from shops group by cube(laptop\_id,provider\_id) order by laptop\_id, provider\_id;





**Разработка процедур на PL/SQL**

1) Функция recommend\_laptops выводит таблицу ноутбуков c количеством amount, у которых цена всё время снижалась.

CREATE OR REPLACE FUNCTION recommend\_laptops(amount int)

returns table (

laptop\_name VARCHAR(50),

producer VARCHAR(25),

price NUMERIC,

characteristic TEXT

)

LANGUAGE PLPGSQL

AS

$$

DECLARE

row record;

error\_text TEXT;

BEGIN

IF amount < 1 THEN

RAISE EXCEPTION 'The amount of laptops cannot be like this %', amount;

END IF;

FOR row IN (SELECT \* FROM laptops ORDER BY price LIMIT amount)

LOOP IF row.price < ALL(SELECT laptops\_audits.price FROM laptops\_audits WHERE laptops\_audits.laptop\_name = row.laptop\_name) THEN

laptop\_name := row.laptop\_name;

producer := UPPER(row.producer);

price := row.price;

characteristic := row.characteristic;

return NEXT;

END IF;

END LOOP;

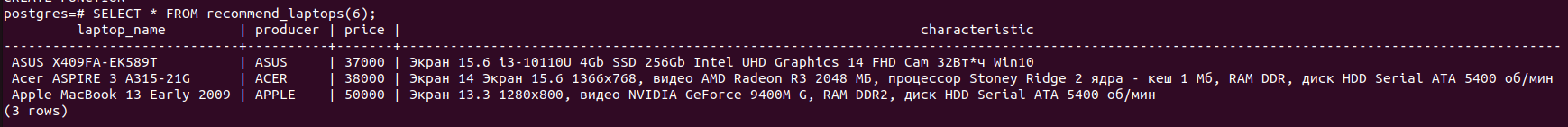
EXCEPTION WHEN OTHERS THEN

GET STACKED DIAGNOSTICS error\_text = MESSAGE\_TEXT;

RAISE EXCEPTION 'Error: %', error\_text;

END;

$$;



2) Процедура transfer\_money переводит количество amount денег продавцу от покупателя по их идентификаторам sender\_id и receiver\_id соответственно.

CREATE OR REPLACE PROCEDURE transfer\_money(

sender\_id int,

receiver\_id int,

amount int

)

LANGUAGE PLPGSQL

AS

$$

DECLARE

old\_sender\_money NUMERIC;

new\_sender\_money NUMERIC;

old\_receiver\_money NUMERIC;

new\_receiver\_money NUMERIC;

error\_text TEXT;

BEGIN

IF amount < 1 THEN

RAISE EXCEPTION 'The amount of money cannot be like this %', amount;

ELSIF sender\_id NOT IN (SELECT customer\_id FROM customers) THEN

RAISE EXCEPTION 'Value % is invalid for sender\_id', sender\_id;

ELSIF receiver\_id NOT IN (SELECT seller\_id FROM sellers) THEN

RAISE EXCEPTION 'Value % is invalid for receiver\_id', receiver\_id;

END IF;

SELECT customer\_balance INTO old\_sender\_money FROM customers WHERE customer\_id = sender\_id;

SELECT seller\_balance INTO old\_receiver\_money FROM sellers WHERE seller\_id = receiver\_id;

IF old\_sender\_money - amount < 0 THEN

RAISE EXCEPTION 'The customer has not enough money for transaction';

END IF;

UPDATE customers SET customer\_balance = customer\_balance - amount WHERE customer\_id = sender\_id;

UPDATE sellers SET seller\_balance = seller\_balance + amount WHERE seller\_id = receiver\_id;

SELECT customer\_balance INTO new\_sender\_money FROM customers WHERE customer\_id = sender\_id;

SELECT seller\_balance INTO new\_receiver\_money FROM sellers WHERE seller\_id = receiver\_id;

IF (old\_sender\_money - new\_sender\_money <> amount) OR (new\_receiver\_money - old\_receiver\_money <> amount) THEN

RAISE EXCEPTION 'The transaction was wrong';

END IF;

EXCEPTION WHEN OTHERS THEN

GET STACKED DIAGNOSTICS error\_text = MESSAGE\_TEXT;

RAISE EXCEPTION 'Error: %', error\_text;

END;

$$

CALL transfer\_money(1, 2, 10000);





3) Триггер laptops\_price\_changes добавляет запись ноутбука в таблицу laptops\_audits, если у ноутбука меняется цена, используя функцию log\_price\_changes.

CREATE OR REPLACE FUNCTION log\_price\_changes()

RETURNS TRIGGER

LANGUAGE PLPGSQL

AS

$$

BEGIN

IF NEW.price <> OLD.price THEN

INSERT INTO laptops\_audits(laptop\_name, producer, price, characteristic)

VALUES(OLD.laptop\_name, OLD.producer, OLD.price, OLD.characteristic);

END IF;

RETURN NEW;

END;

$$

CREATE TRIGGER laptops\_price\_changes

BEFORE UPDATE

ON laptops

FOR EACH ROW

EXECUTE PROCEDURE log\_price\_changes();



**Список использованных источников**

1. Документация PostgreSQL https://www.postgresql.org/docs
2. PostgreSQL с примерами [https://www.postgresqltutorial.com/postgresql-plpgsql](https://vk.com/away.php?to=https%3A%2F%2Fwww.postgresqltutorial.com%2Fpostgresql-plpgsql&cc_key=)