Ртищева Алена Сергеевна
Группа М08-501НД
Науки о данных

Домашнее задание 1. Создание и нормализация базы данных

Дана база данных **customer_and_transaction.xlsx**

В ней содержатся две таблицы:
1. **transaction**
2. **customer**

Таблица **transaction** содержит следующие колонки (слева укажем предполагаемый тип данных в синтаксисе dbdiagram.io / не PostgreSQL, хотя при импорте, по-видимому, большинство данных придется задать типом varchar!!!):

1. transaction_id – integer [primary key]
2. product_id – integer [not null]
3. customer_id – integer [not null]
4. transaction_date – timestamp [not null]
5. online_order – boolean
6. order_status – varchar(50) [not null]
7. brand – varchar(100) [not null]
8. product_line – varchar(50)
9. product_class – varchar(50)
10. product_size – varchar(50)
11. list_price – decimal(10,2) [not null]
12. standard_cost – decimal(10,2)

Таблица **customer** содержит следующие колонки:

1. customer_id – integer [primary key]
2. first_name – varchar(100) [not null]
3. last_name – varchar(100) [not null]
4. DOB – date
5. job_title – varchar(100)
6. job_industry_category – varchar(100)
7. wealth_segment – varchar(50)
8. deceased_indicator – varchar(10) [not null]
9. owns_car – varchar(10)
10. address – text
11. postcode – varchar(20)
12. state – varchar(50)
13. country – varchar(50)
14. property_valuation – integer

Анализ таблиц

Таблица **transaction**:

transaction_id – первичный ключ (уникальный идентификатор транзакции).

Функциональные зависимости от transaction_id:

transaction_id → (product_id,
customer_id,

transaction_date,
online_order,
order_status,
brand,
product_line,
product_class,
product_size,
list_price,
standard_cost)

Частичные зависимости:

product_id → (brand,
product_line,
product_class,
product_size,
list_price,
standard_cost)

customer_id → (все атрибуты из таблицы customer)

Таблица **customer**:

customer_id – первичный ключ (уникальный идентификатор покупателя).

Функциональные зависимости:

customer_id → (first_name,
last_name,
DOB,
job_title,
job_industry_category,
wealth_segment,
deceased_indicator,
owns_car,
address,
postcode)

Эта база данных имеет 1НФ (первую нормальную форму) потому что:

1. Все атрибуты атомарны (каждое поле содержит только одно значение).
2. Нет повторяющихся групп (в таблицах нет столбцов, которые повторяют одну и ту же информацию в разных колонках).
3. Есть первичный ключ.

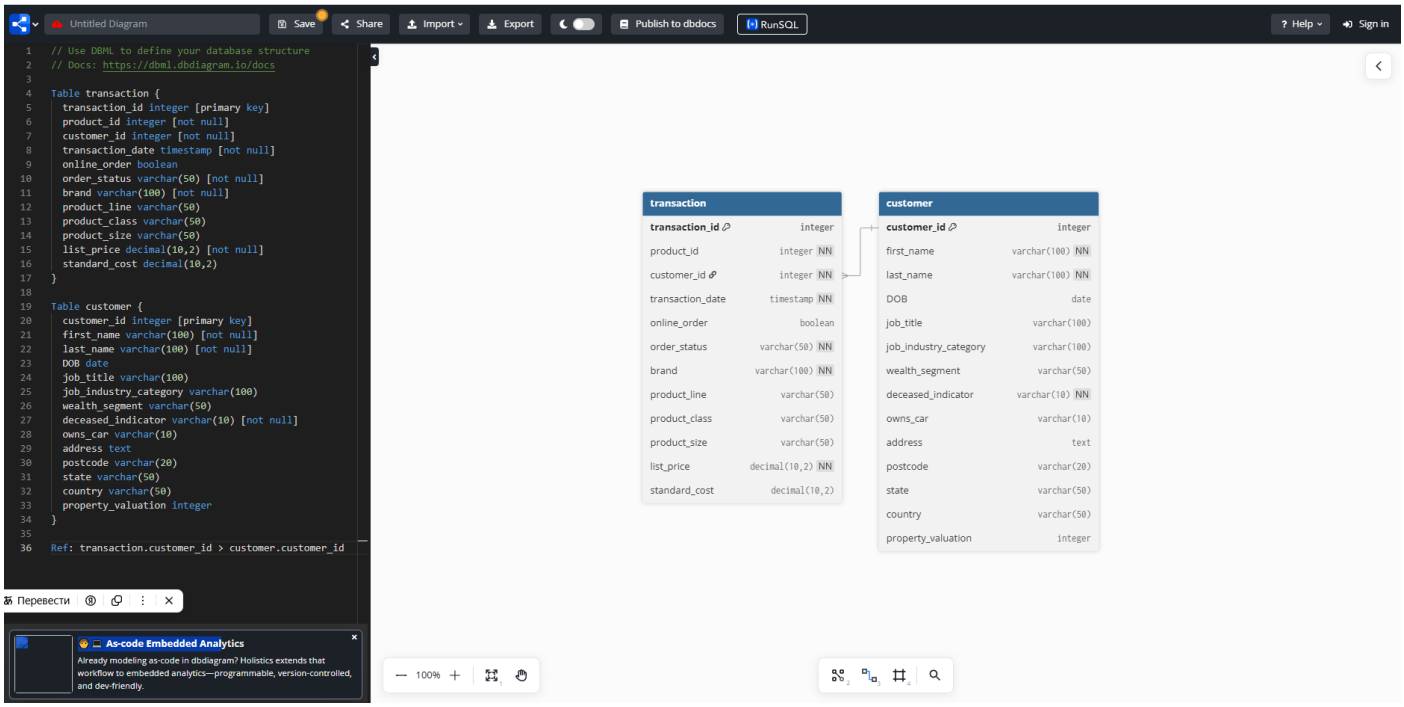


Рисунок 1 – 1НФ (исходный вариант)

Приведем данные к 2НФ (второй нормальной форме).

Мы выделили в таблице **transaction** зависимость:

product_id → (brand,
product_line,
product_class,
product_size,
list_price,
standard_cost)

Преобразуем таблицу **transaction** в две таблицы **transaction** и **product**. Таблицу **customer** оставим как есть.

Таблица **transaction** содержит следующие колонки:

1. transaction_id – integer [primary key]
2. product_id – integer [not null]
3. customer_id – integer [not null]
4. transaction_date – timestamp [not null]
5. online_order – boolean
6. order_status – varchar(50) [not null]

Таблица **product** содержит следующие колонки:

1. product_id – integer [primary key]
2. brand – varchar(100) [not null]
3. product_line – varchar(50)
4. product_class – varchar(50)
5. product_size – varchar(50)
6. list_price – decimal(10,2) [not null]
7. standard_cost – decimal(10,2)

Таблица **customer** содержит следующие колонки:

1. customer_id – integer [primary key]
2. first_name – varchar(100) [not null]
3. last_name – varchar(100) [not null]

4. DOB – date
5. job_title – varchar(100)
6. job_industry_category – varchar(100)
7. wealth_segment – varchar(50)
8. deceased_indicator – varchar(10) [not null]
9. owns_car – varchar(10)
10. address – text
11. postcode – varchar(20)
12. state – varchar(50)
13. country – varchar(50)
14. property_valuation – integer

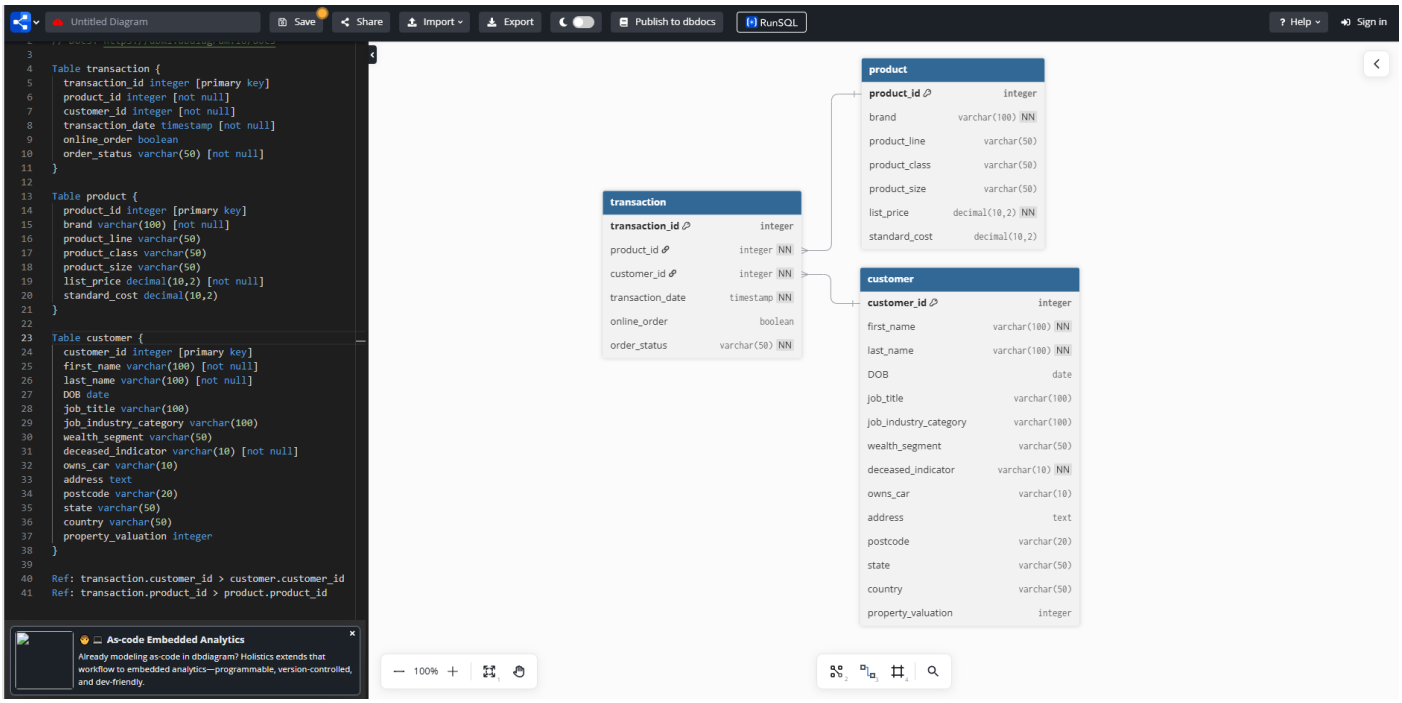


Рисунок 2 – 2НФ

Эта база данных имеет 2НФ (вторую нормальную форму) потому что:

1. Соотвествует всем требованиям 1НФ (все атрибуты атомарны, нет повторяющихся групп, есть первичный ключ)
2. Отсутствуют частичные зависимости:
   product_id → (brand, product_line, product_class, product_size, list_price, standard_cost)
3. Каждый неключевой атрибут зависит от первичного ключа:
   В **transaction**: все атрибуты зависят от transaction_id
   В **customer**: все атрибуты зависят от customer_id
   В **product**: все атрибуты зависят от product_id

Приведем данные к 3НФ (третьей нормальной форме).

Для этого нужно установить транзитивные зависимости.

В таблице **customer** можно установить следующие зависимости:

job_title – job_industry_category (относятся к работе покупателя)
wealth_segment – property_valuation (относятся к благосостоянию покупателя)
address – postcode (относятся к адресу проживания покупателя)
state – country (относятся к месту проживания покупателя)

Преобразуем таблицу customer в четыре таблицы: **customer**, **job_industry**, **wealth_segment** и **address**.

Таблицы **transaction** и **product** не меняем.

Таблица **transaction** содержит следующие колонки:

1. transaction_id – integer [primary key]
2. product_id – integer [not null]
3. customer_id – integer [not null]
4. transaction_date – timestamp [not null]
5. online_order – boolean
6. order_status – varchar(50) [not null]

Таблица **product** содержит следующие колонки:

1. product_id – integer [primary key]
2. brand – varchar(100) [not null]
3. product_line – varchar(50)
4. product_class – varchar(50)
5. product_size – varchar(50)
6. list_price – decimal(10,2) [not null]
7. standard_cost – decimal(10,2)

Таблица **customer** содержит следующие колонки:

1. customer_id – integer [primary key]
2. first_name – varchar(100) [not null]
3. last_name – varchar(100) [not null]
4. DOB – date
6. job_industry_category_id – integer [not null]
7. wealth_segment_id – integer [not null]
8. deceased_indicator – varchar(10) [not null]
9. owns_car – varchar(10)
10. address_id – integer [not null]

Таблица **job_industry** содержит следующие колонки:

1. job_industry_category_id – integer [primary key]
2. industry_name – varchar(100) [not null]
3. information – text

Таблица **wealth_segment** содержит следующие колонки:

1. wealth_segment_id – integer [primary key]
2. segment_name – varchar(50) [not null]
3. min_valuation – integer [not null]
4. max_valuation – integer [not null]

Таблица **address** содержит следующие колонки:

1. address_id integer – [primary key]
2. address_1 – text [not null]
3. address_2 – text
4. postcode – varchar(20) [not null]
5. state – varchar(50) [not null]
6. country – varchar(50) not null

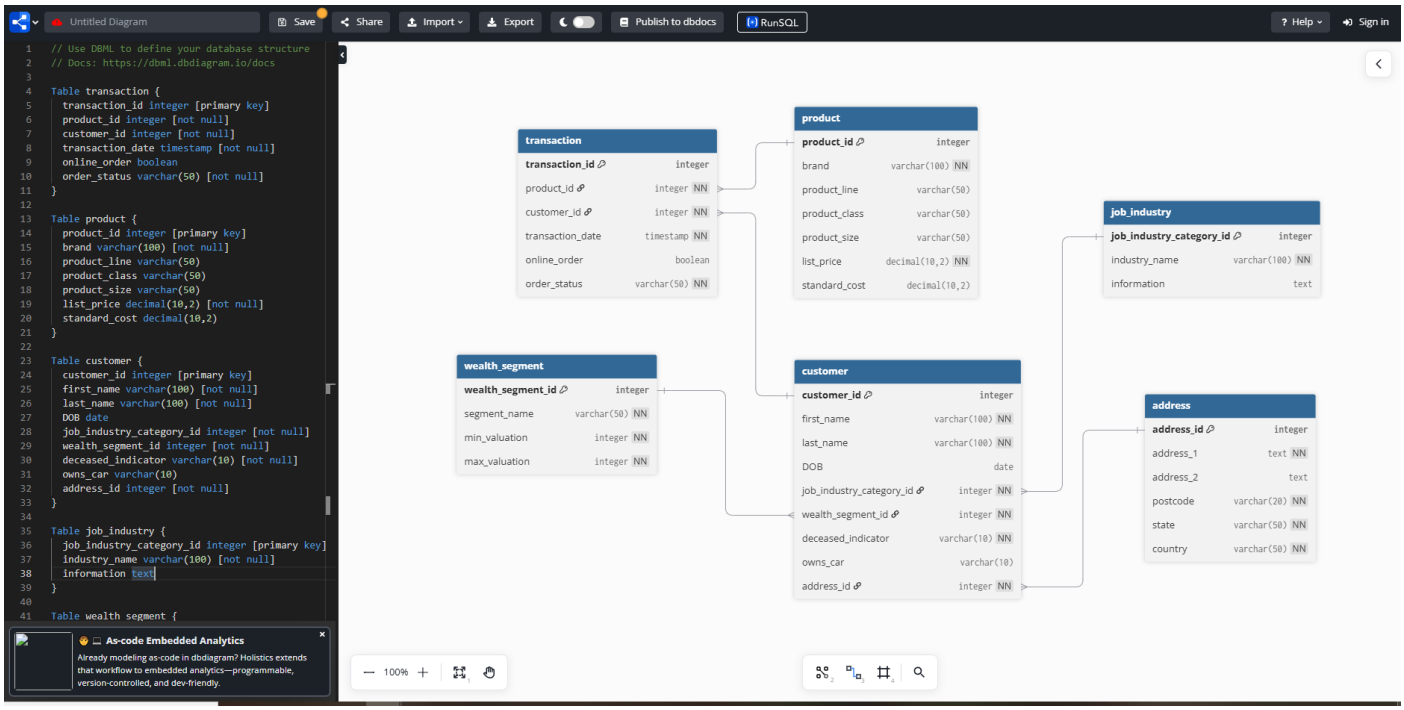
Эта база данных имеет 3НФ (третью нормальную форму) потому что:

1. Соотвествует всем требованиям 1НФ и 2НФ (все атрибуты атомарны, нет повторяющихся групп, есть первичный ключ, нет частичных зависимостей)

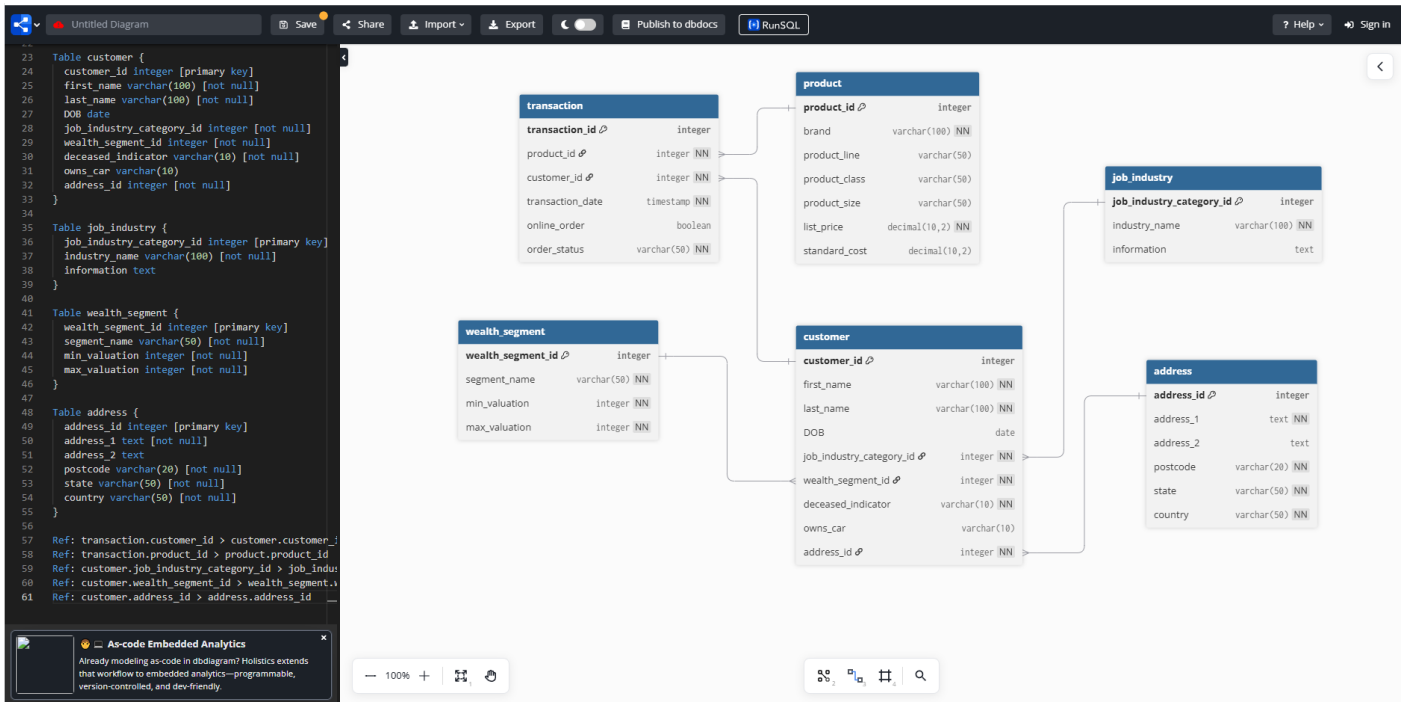
2. Отсутствуют транзитивные зависимости:

job_title - job_industry_category (относятся к работе покупателя);
wealth_segment - property_valuation (относятся к благосостоянию покупателя);
address - postcode (относятся к адресу проживания покупателя);
state - country (относятся к месту проживанию покупателя).

3. Каждый неключевой атрибут зависит только от первичного ключа:
   В **transaction**: все атрибуты зависят только от transaction_id
   В **customer**: все атрибуты зависят только от customer_id
   В **product**: все атрибуты зависят только от product_id
   В **job_industry**: все атрибуты зависят только от job_industry_category_id
   В **wealth_segment**: все атрибуты зависят только от wealth_segment_id
   В **address**: все атрибуты зависят только от address_id



а) верхняя часть левого окна



б) нижняя часть левого окна

Рисунок 3 – 3НФ

Далее работаем с базой данных в программе DBeaver 25.2.4.

Для начала сохраняем базу данных **customer_and_transaction.xlsx** в виде двух фалов: **customer.csv** и **transaction.csv**.

Создаем две таблицы **transaction_old** и **customer_old** с колонками, соответствующим 1НФ, но тип данных INTEGER присваиваем только transaction_id и customer_id, все остальные будут типом VARCHAR или TEXT.

Эти таблицы нужны только для того, чтобы импортировать данные.

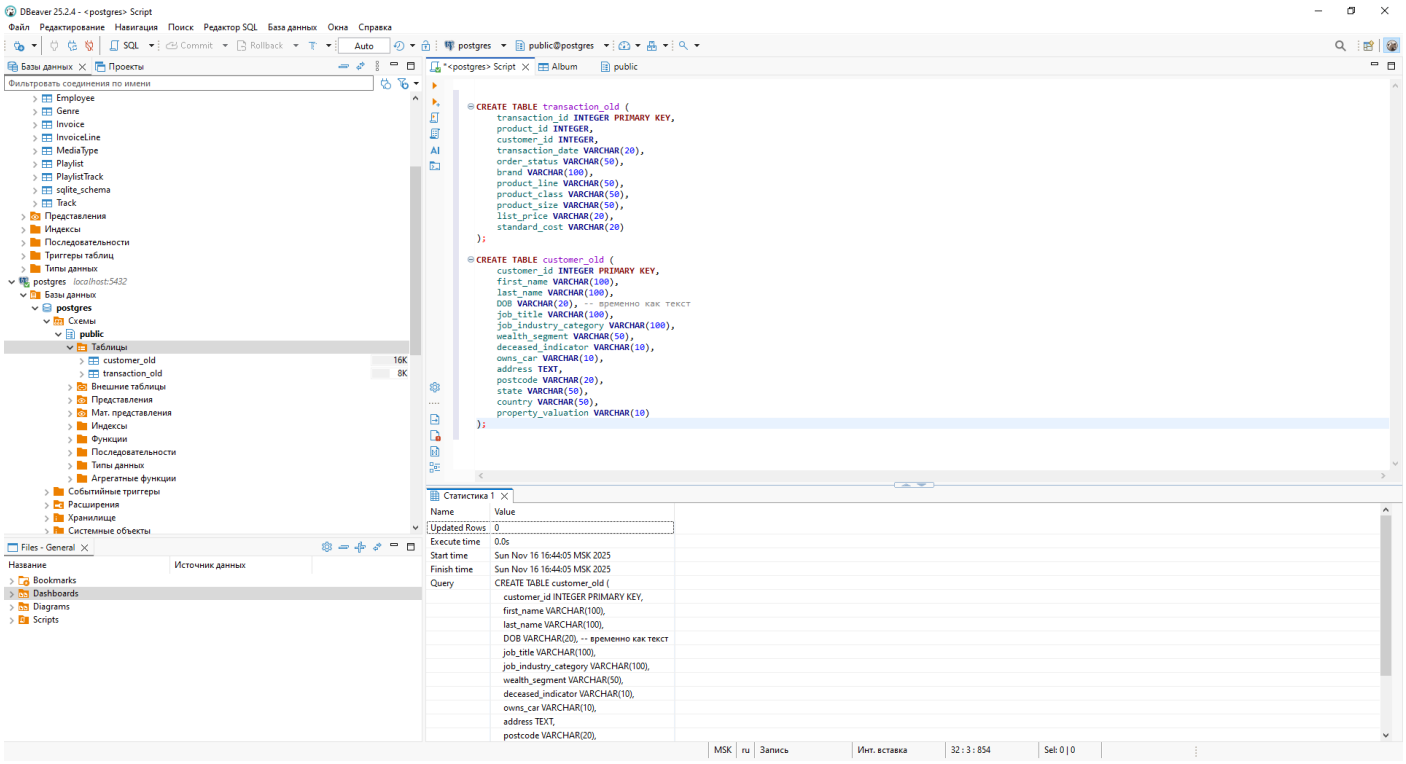


Рисунок 4 – Создание таблиц **transaction_old** и **customer_old**

Далее импортируем данные из **customer.csv** и **transaction.csv**.

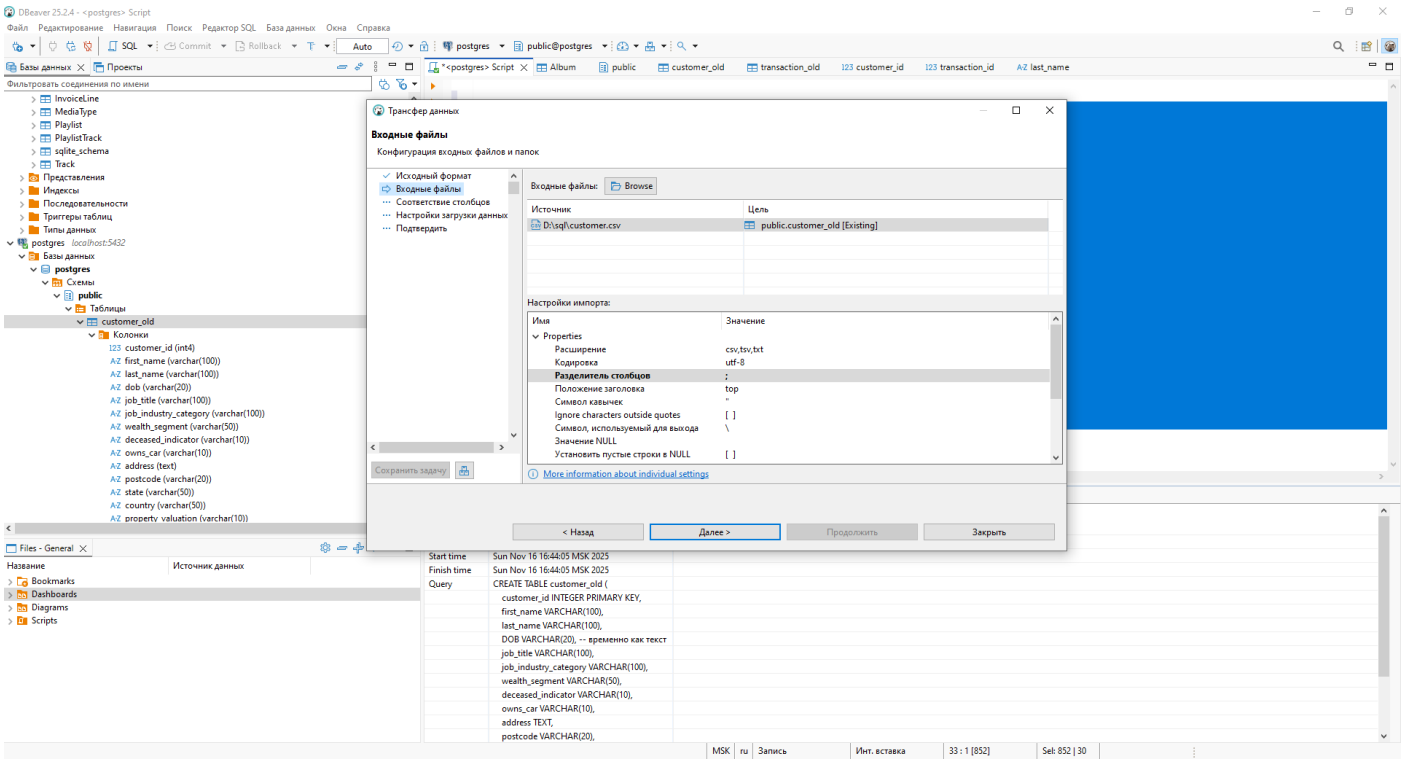


Рисунок 5 – Импорт данных в таблицу **customer_old**

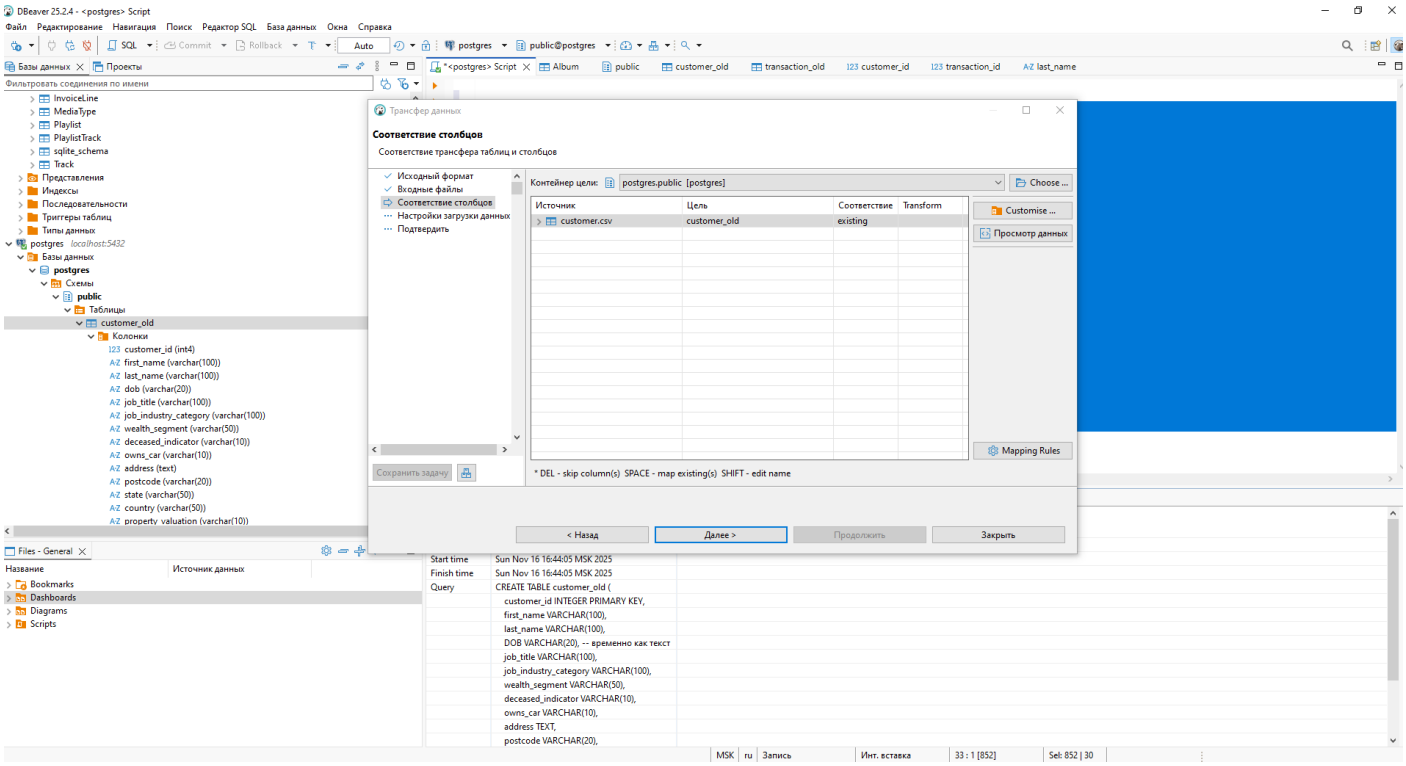


Рисунок 6 – Импорт данных в таблицу **customer_old**

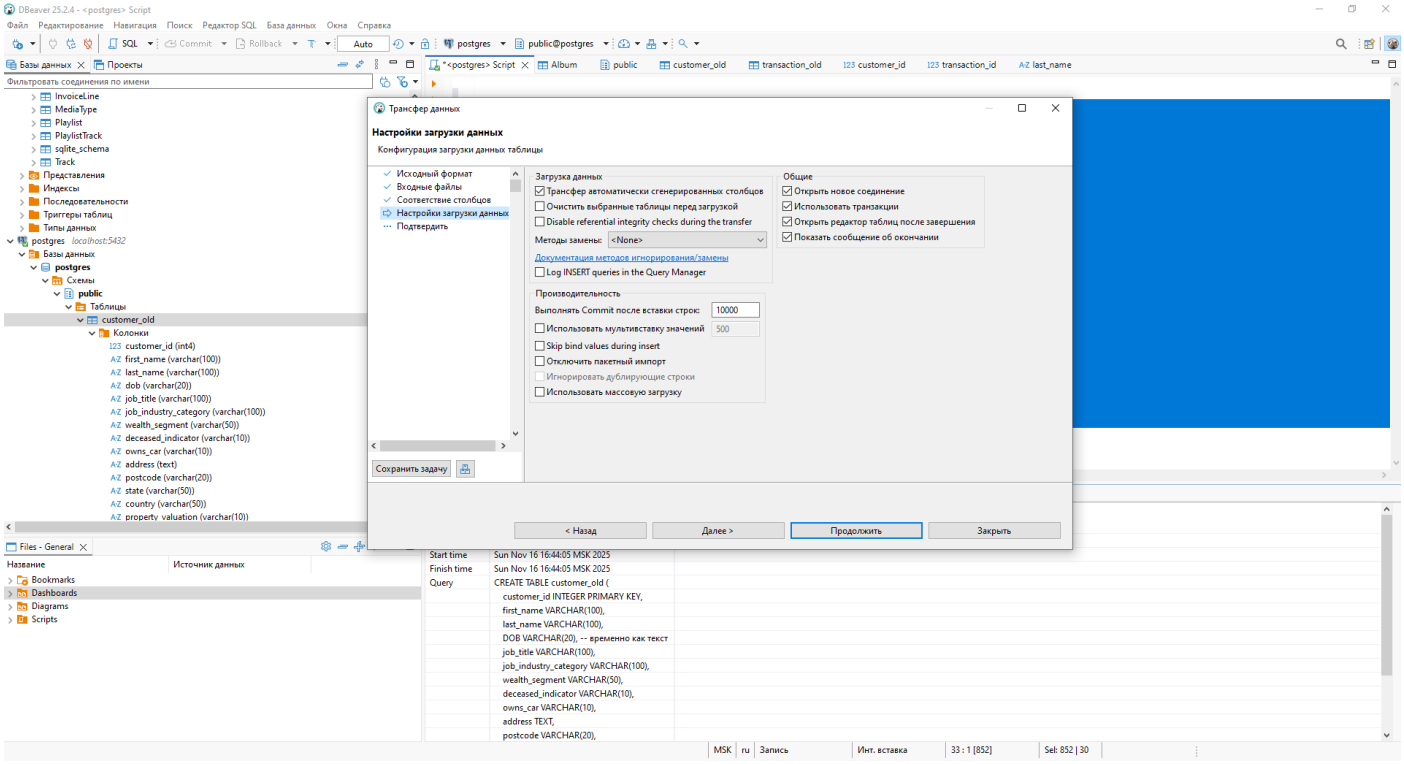


Рисунок 7 – Импорт данных в таблицу **customer_old**

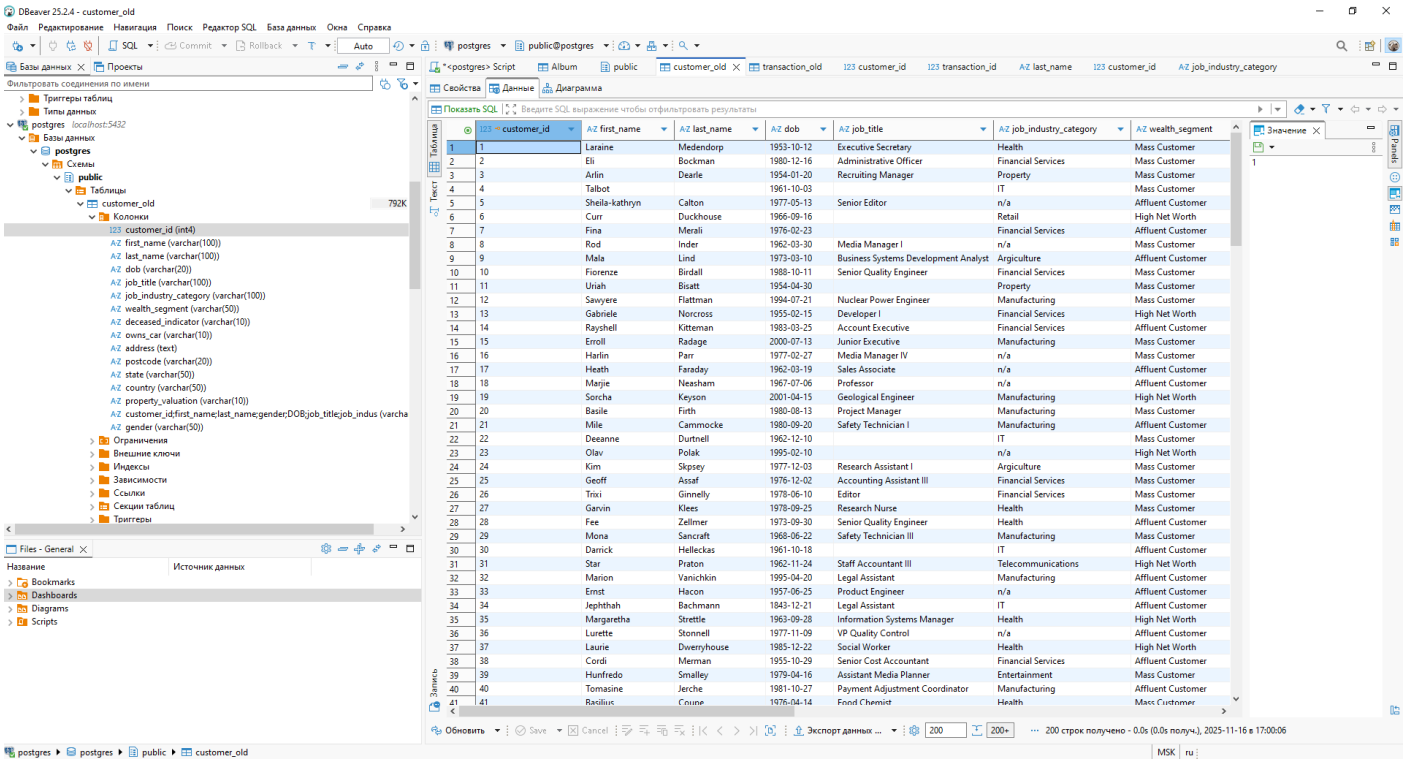


Рисунок 8 – Просмотр данных таблицы **customer_old**

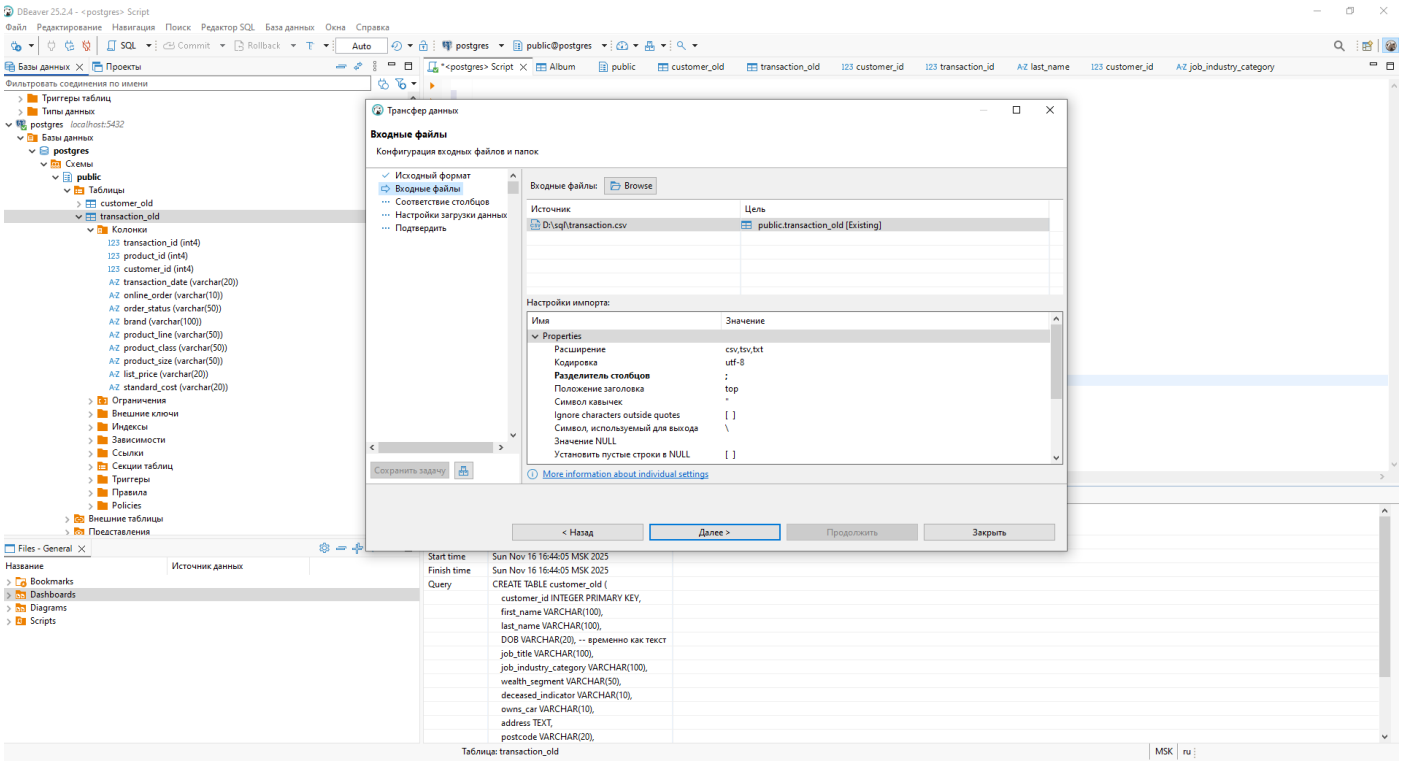


Рисунок 9 – Импорт данных в таблицу **transaction_old**

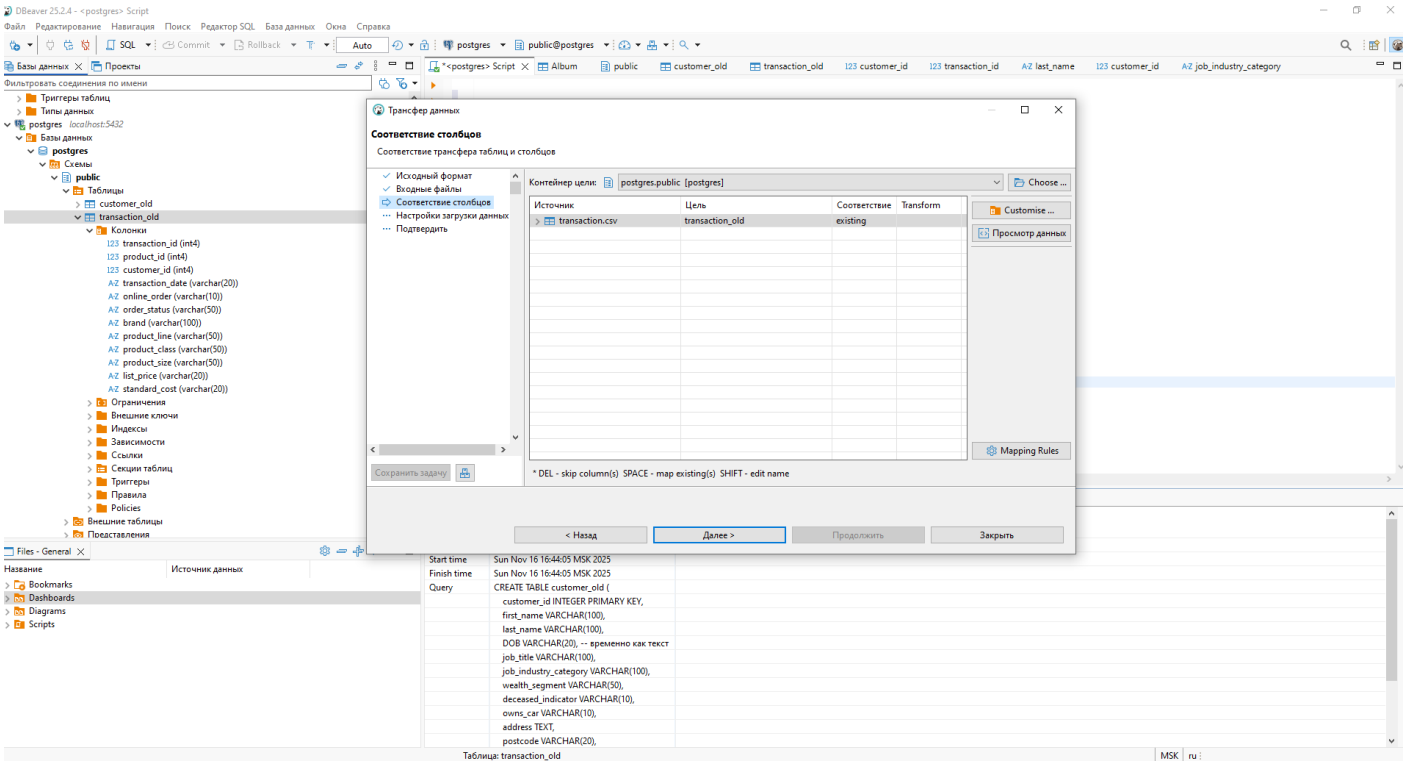


Рисунок 10 – Импорт данных в таблицу **transaction_old**

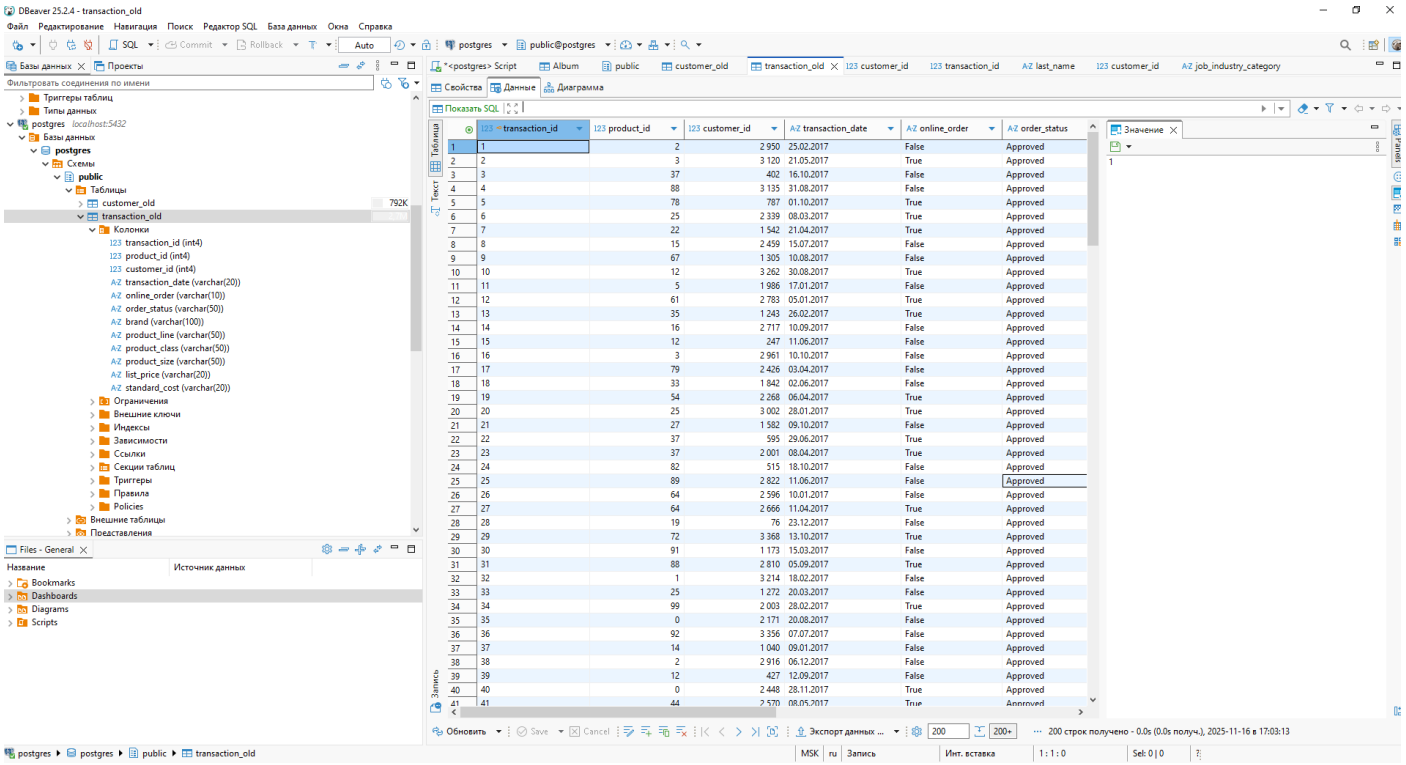


Рисунок 11 – Просмотр данных таблицы **transaction_old**

Создаем таблицы: **job_industry**, **wealth_segment**, **address**, **product**, **customer**, **transaction** с колонками и типами данных, соответствующими 3НФ.

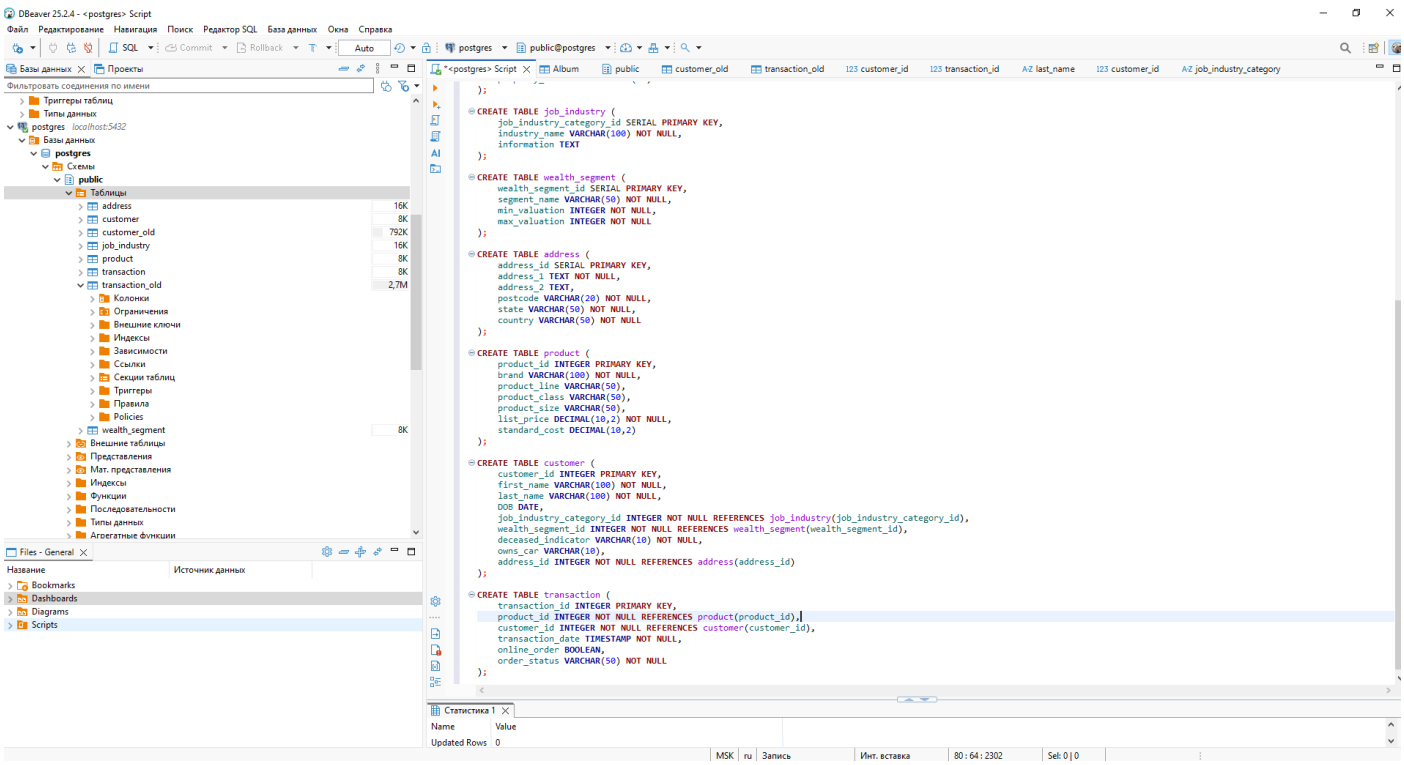


Рисунок 12 – Создание таблиц **job_industry**, **wealth_segment**, **address**, **product**, **customer**, **transaction**

Заполняем таблицу **job_industry**, используя данные из **customer_old**, при этом берем только уникальные значения job_industry_category, а в колонке information пишим “Industry category from customer data”. Хотя эту колонку можно было не создавать, но так пишут в примерах и в принципе она может быть полезной.

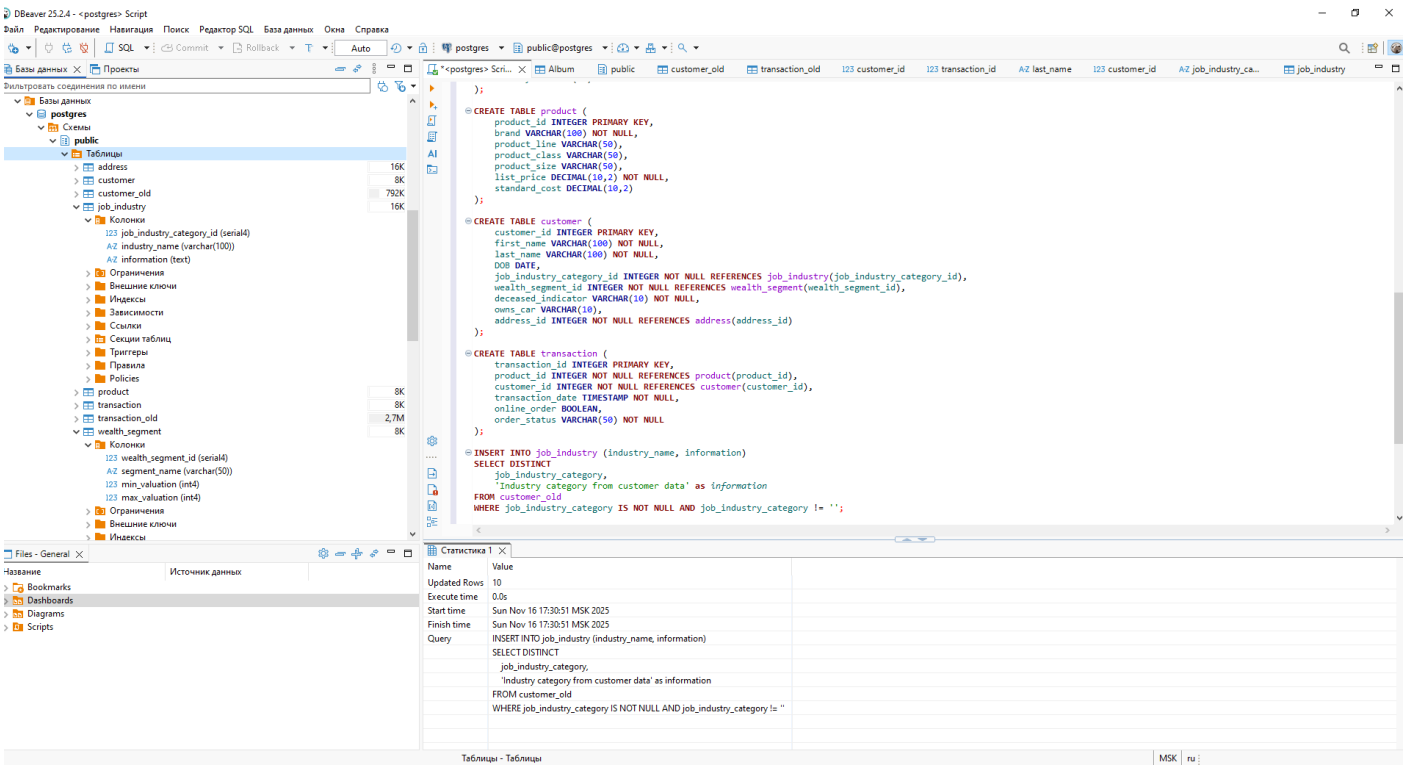


Рисунок 14 – Заполнение таблицы **job_industry**

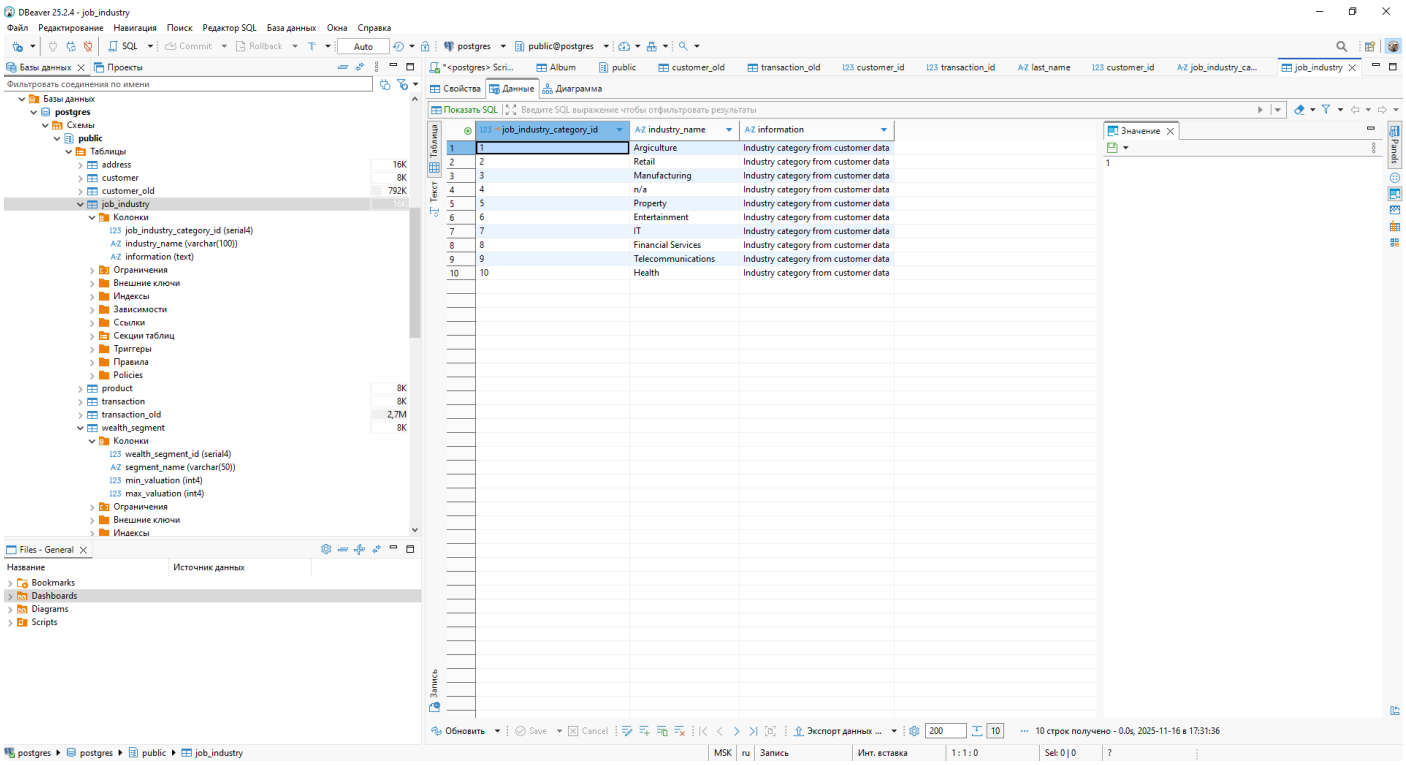


Рисунок 13 – Просмотр данных таблицы **job_industry**

Заполняем таблицу **wealth_segment**, используя данные из **customer_old**, при этом берем только уникальные поля wealth_segment. А далее рассматриваем минимальную и максимальную оценку собственности для каждого сегмента. В property_valuation были нечисловые значения, поэтому код преобразует эти данные в тип INTEGER, исключает NULL, пустые строки и нечисловые данные, группирует по сегментам богатства, рассчитывает фактический диапазон значений для каждого сегмента.

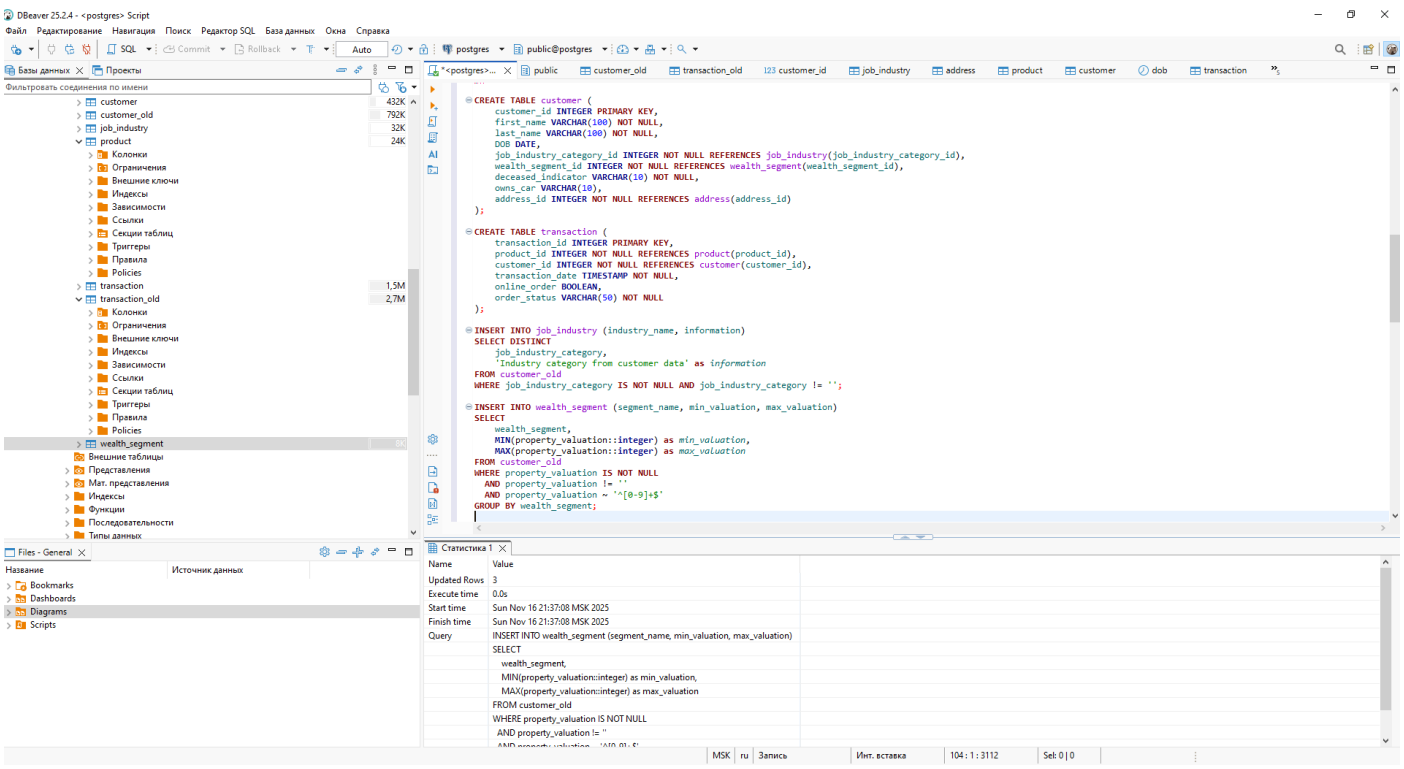


Рисунок 15 – Заполнение таблицы **wealth_segment**

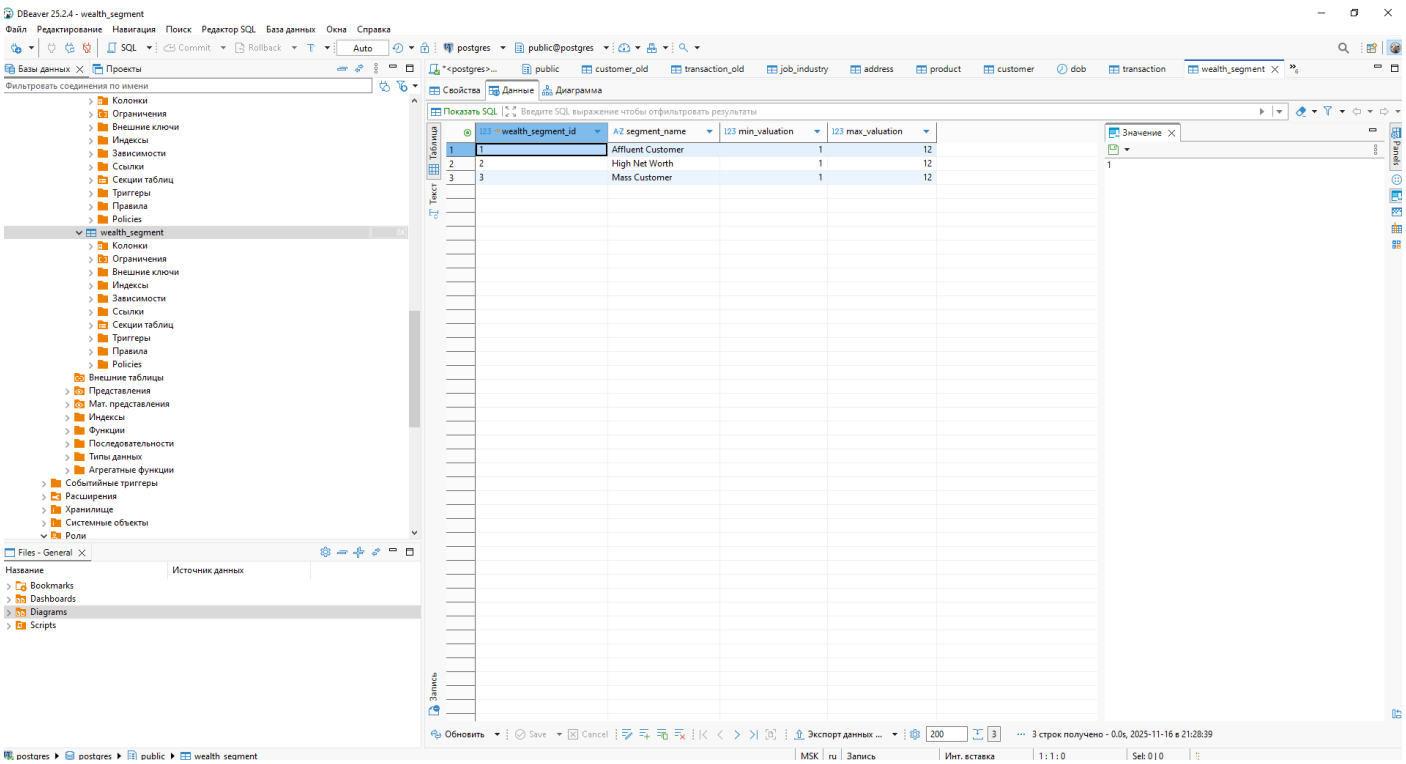


Рисунок 16 – Просмотр данных таблицы **wealth_segment**

Заполняем таблицу **address**, используя данные из **customer_old**, при этом выбираем только уникальные адреса и вставляем их в колонку address_1. Колонка address_1 пустая (создана для удобства), postcode, state, country копируются как есть. Фильтруем записи и вставляем те адреса, которые не NULL.

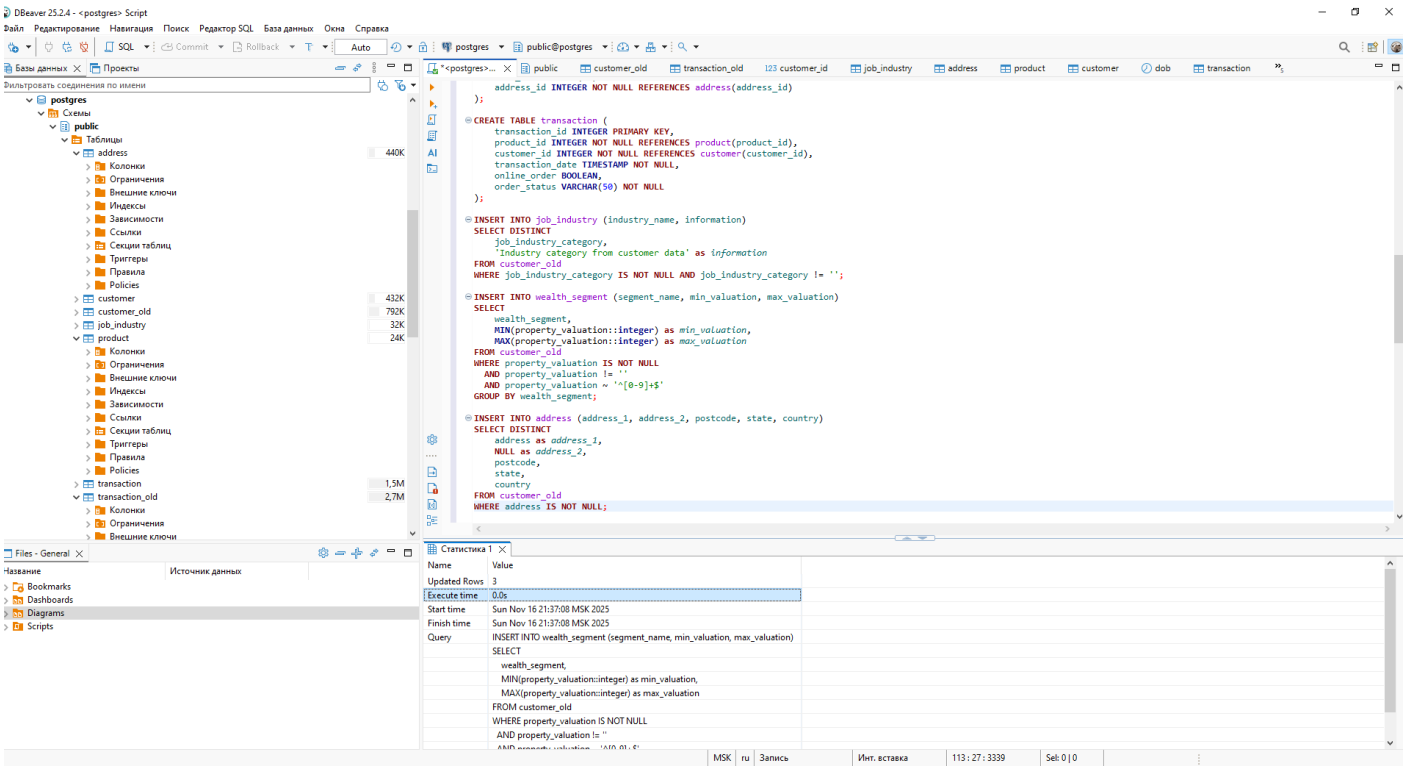


Рисунок 17 – Заполнение таблицы **address**

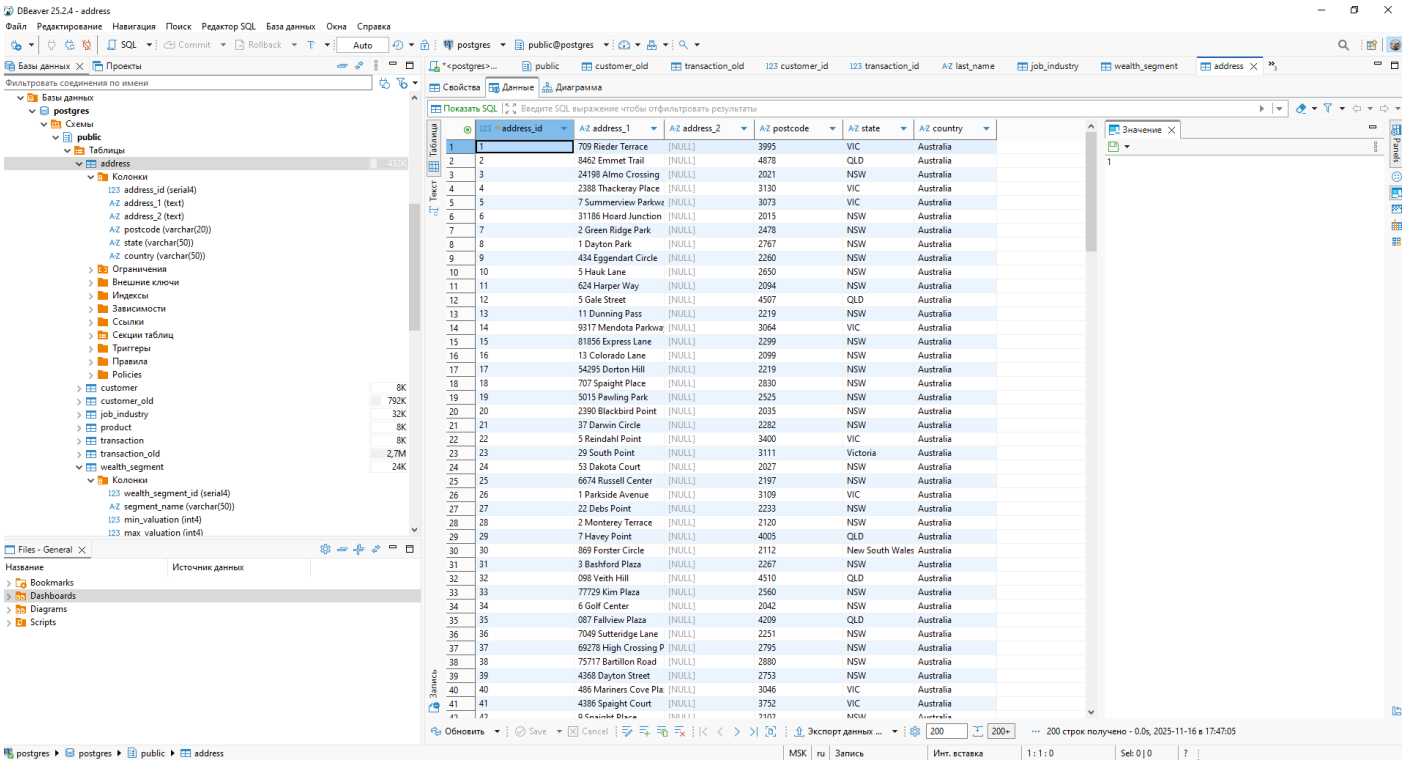


Рисунок 18 – Просмотр данных таблицы **address**

Заполняем таблицу **product**, используя данные из **transaction_old**, при этом код выбирает по одной уникальной записи для каждого product_id (при дубликатах берется версия с наибольшей ценой), очищает и преобразует текстовые значения цен в числовой формат с двумя знаками после запятой, заменяя запятые на точки (в примерах, часто запятой отделяют разряды, а не дроби) и обрабатывая пустые значения, а также исключает записи без product_id.

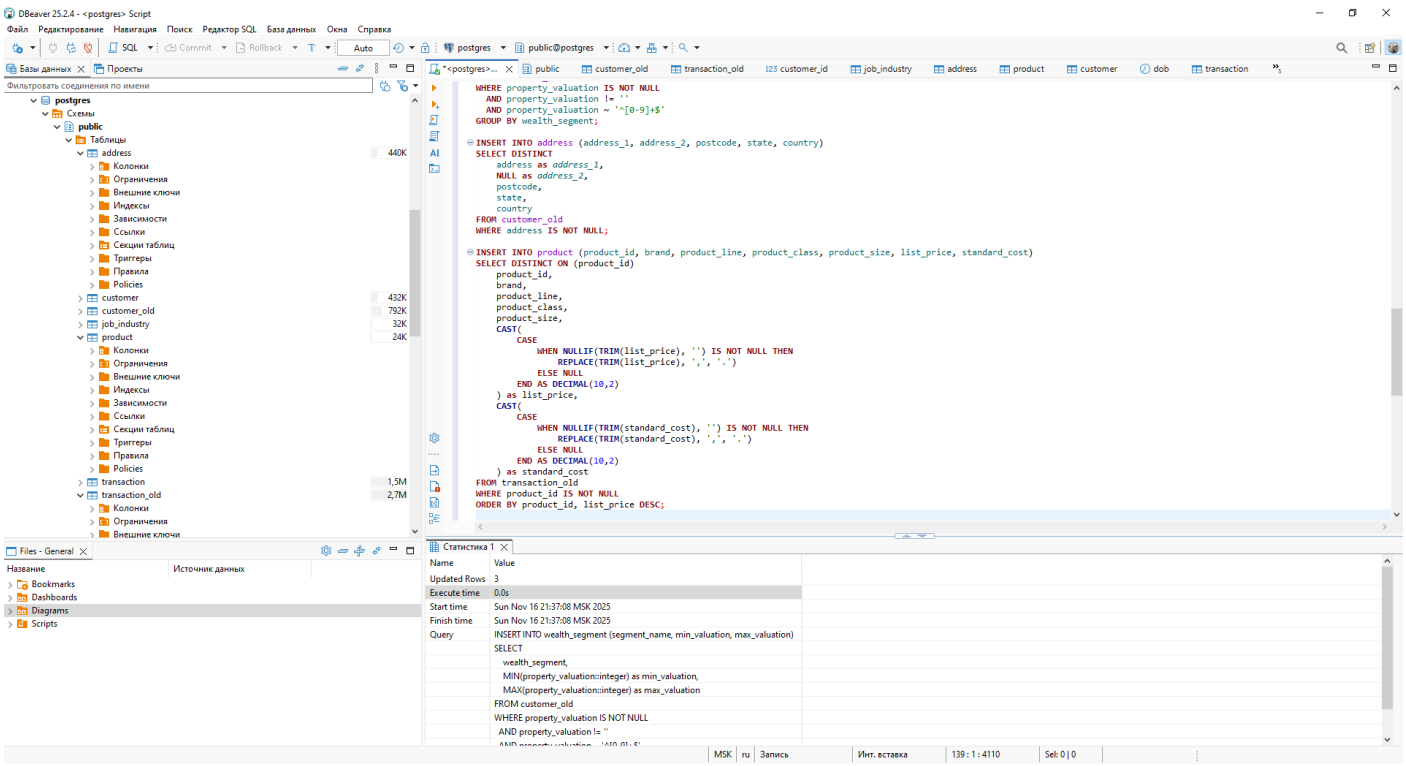


Рисунок 19 – Заполнение таблицы **product**

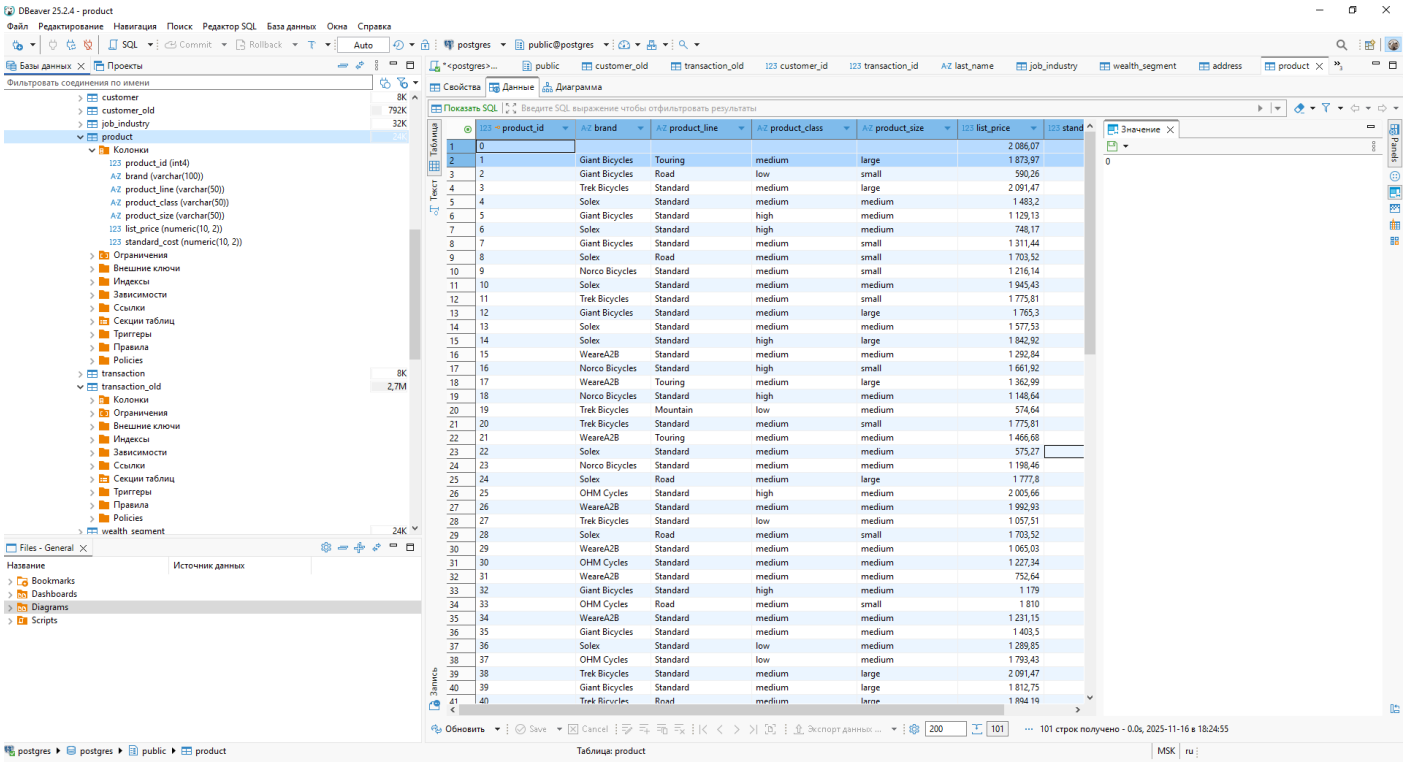


Рисунок 20 – Просмотр данных таблицы **product**

Заполняем таблицу **customer**, используя данные из **customer_old**. Код связывает данные клиентов с соответствующими справочными таблицами через LEFT JOIN: для job_industry_category_id связывается с таблицей job_industry по названию индустрии, для wealth_segment_id – с таблицей wealth_segment по названию сегмента богатства, а для address_id – с таблицей address по совпадению адреса и почтового индекса. Особое внимание уделяется преобразованию даты рождения (DOB) – проверяется корректность формата “YYYY-MM-DD” перед преобразованием в тип DATE, некорректные значения заменяются на NULL.

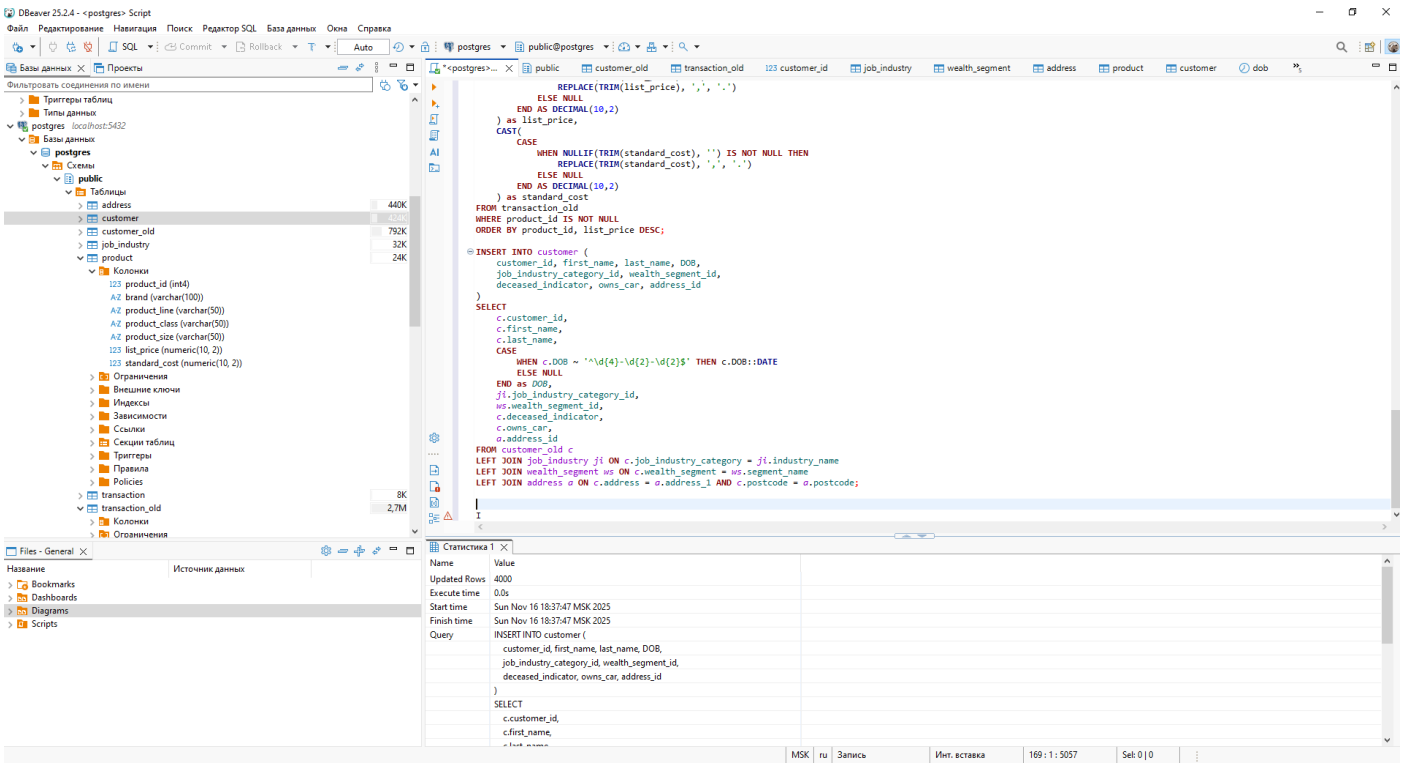


Рисунок 21 – Заполнение таблицы **customer**

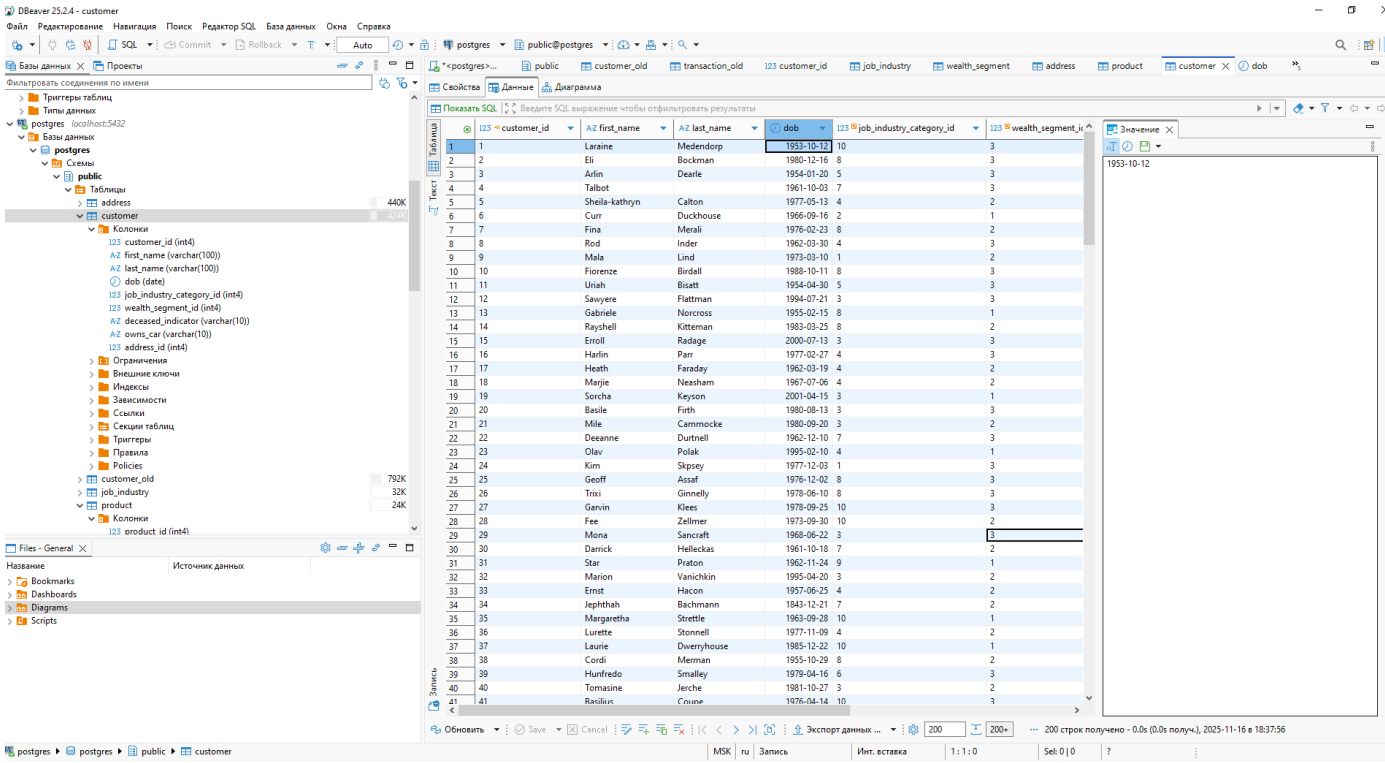


Рисунок 22 – Просмотр данных таблицы **customer**

Заполняем таблицу **transaction**, используя данные из **transaction_old**. Изначально в колонке transaction_date предполагалось ограничение NOT NULL. Но оказалось, что есть записи с некорректными данными. Их решаем оставить и снять ограничение с этой колонки.

Затем данные переносятся с преобразованием форматов. Проверяется корректность даты транзакции в формате “YYYY-MM-DD HH:MM:SS” и преобразуется в TIMESTAMP, некорректные значения заменяются на NULL; значения online_order преобразуются из текстовых “True” / “False” в булевы true / false, а пустые строки и NULL остаются NULL; order_status переносится как есть. Фильтрация записей через WHERE гарантирует, что вставляются только те транзакции, у которых customer_id и product_id существуют в соответствующих таблицах **customer** и **product**, обеспечивая целостность ссылок и соблюдение внешних ключей в новой структуре базы данных.

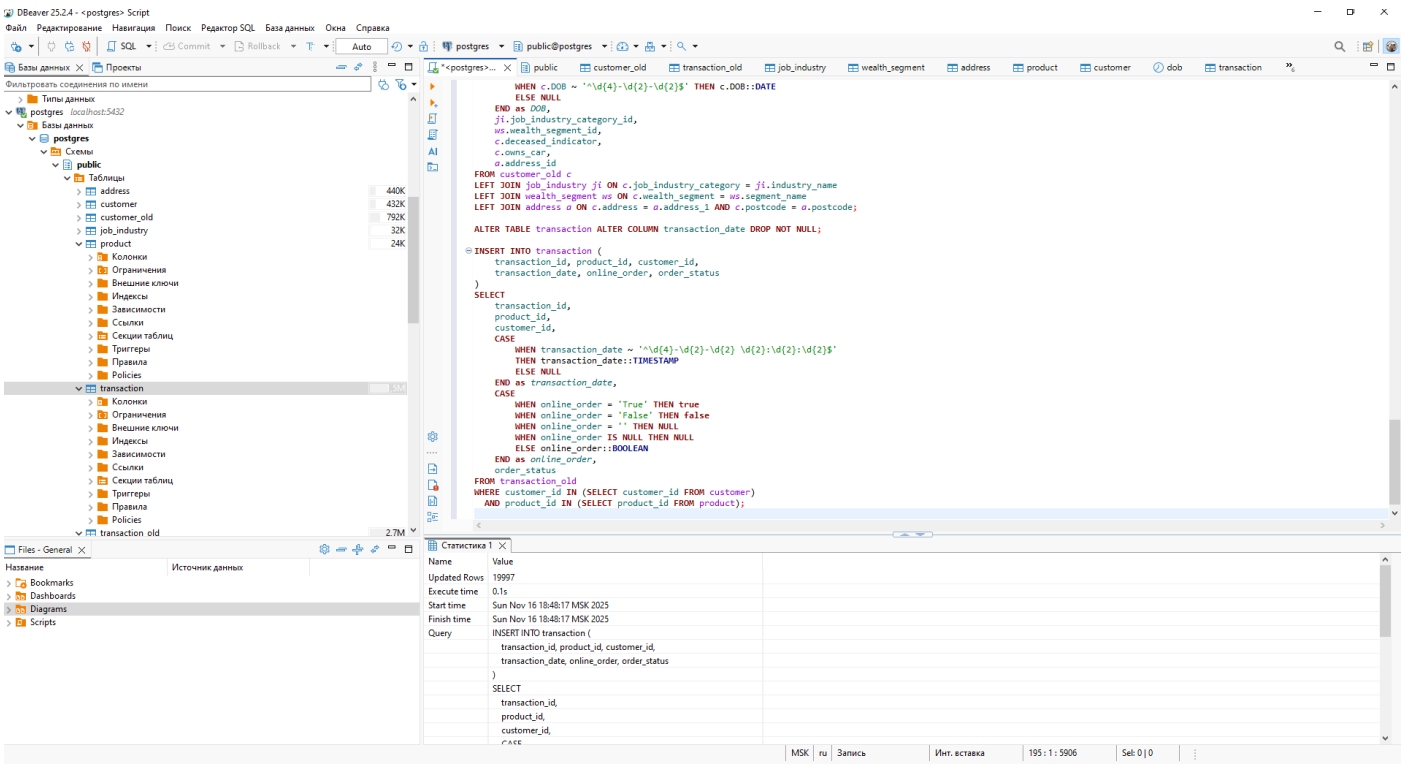


Рисунок 23 – Заполнение таблицы **transaction**

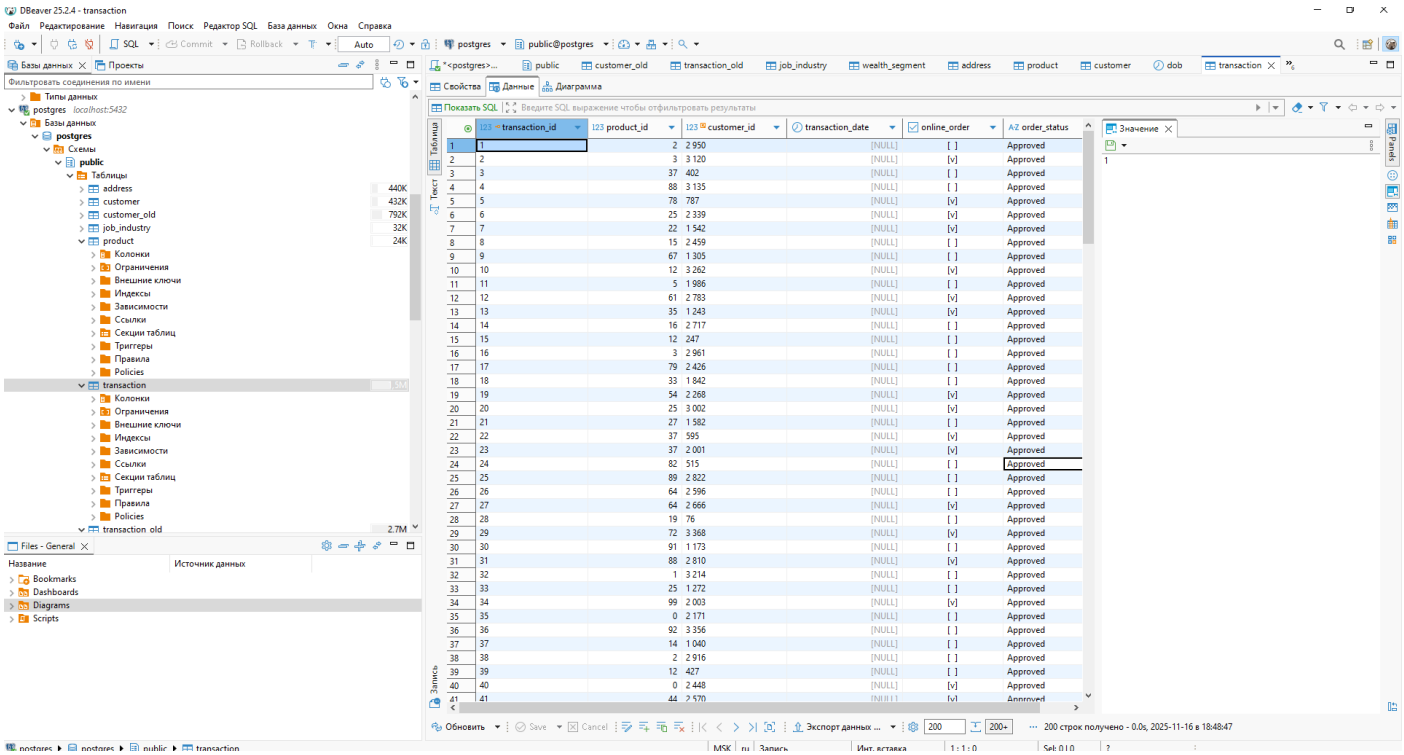


Рисунок 24 – Просмотр данных таблицы **transaction**

Преобразованную до 3НФ базу данных в формате .csv экспортированную из таблиц: **job_industry**, **wealth_segment**, **address**, **product**, **customer**, **transaction** можно найти в папке “ Экспортируемые итоговые csv”.