

## 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.

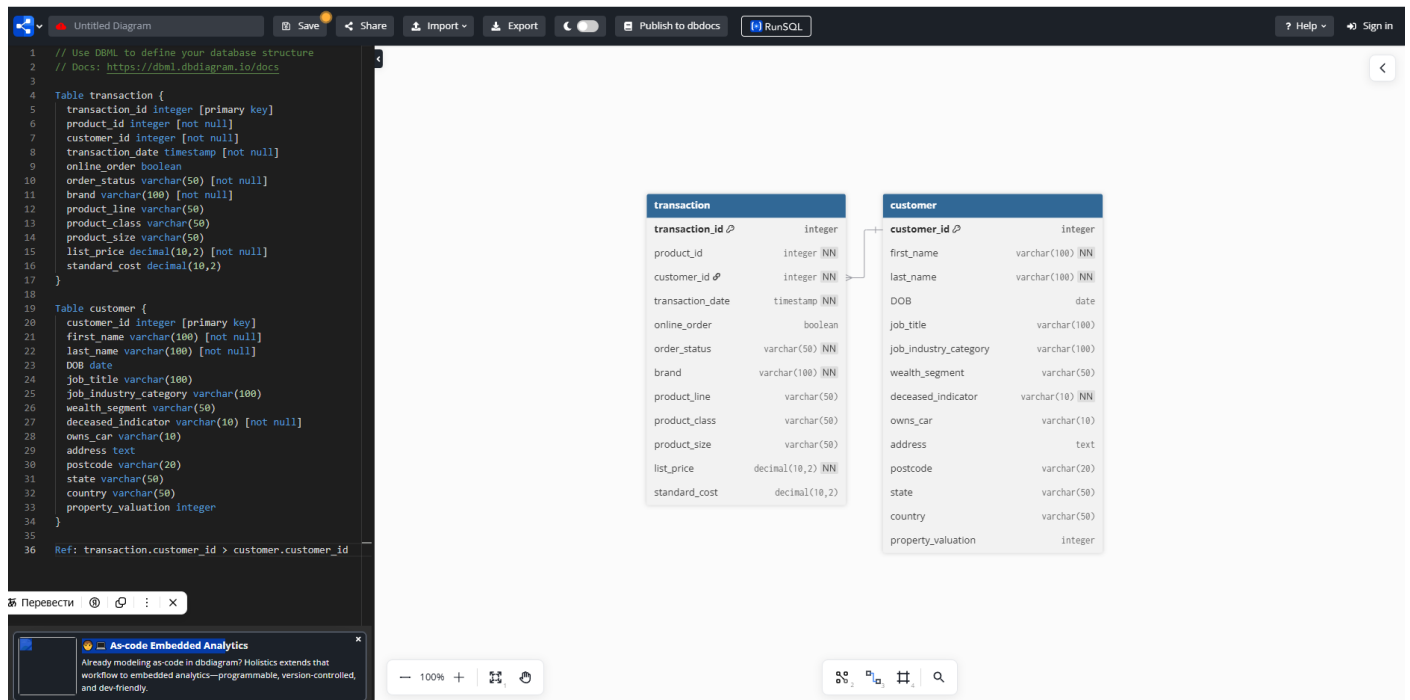


Рисунок 1 – 1НФ (исходный вариант)

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

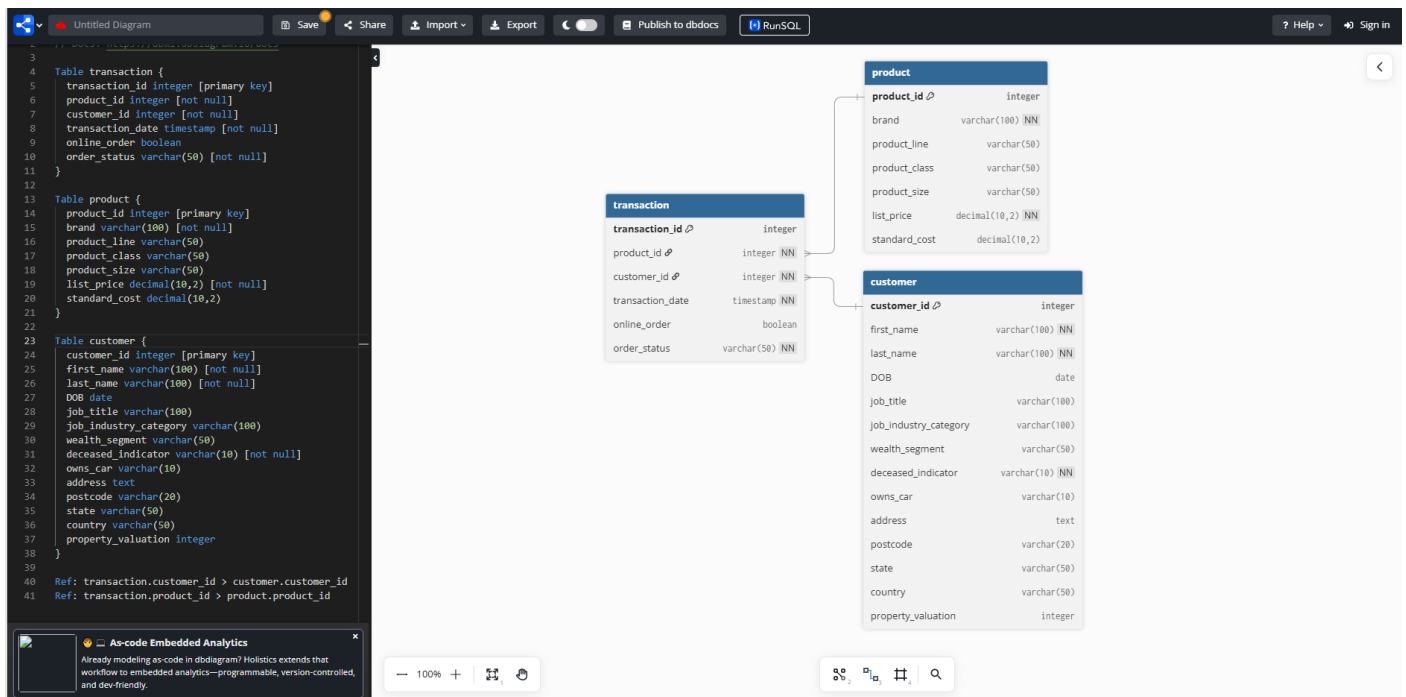


Рисунок 2 – 2НФ

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

1. Соответствует всем требованиям 1НФ (все атрибуты атомарны, нет повторяющихся групп, есть первичный ключ)
2. Отсутствуют частичные зависимости:  
 $product\_id \rightarrow (brand, product\_line, product\_class, product\_size, list\_price, standard\_cost)$
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. Соответствует 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

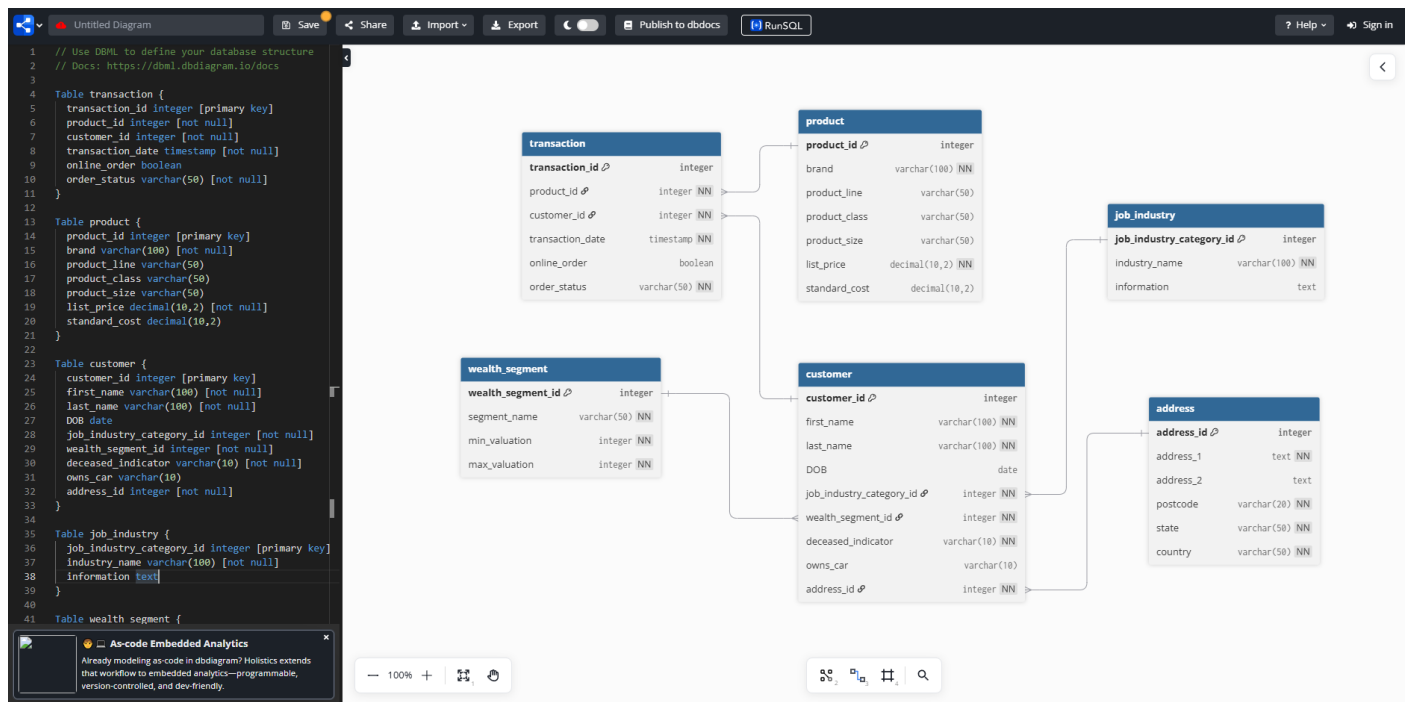
В **customer**: все атрибуты зависят только от customer\_id

В **product**: все атрибуты зависят только от product\_id

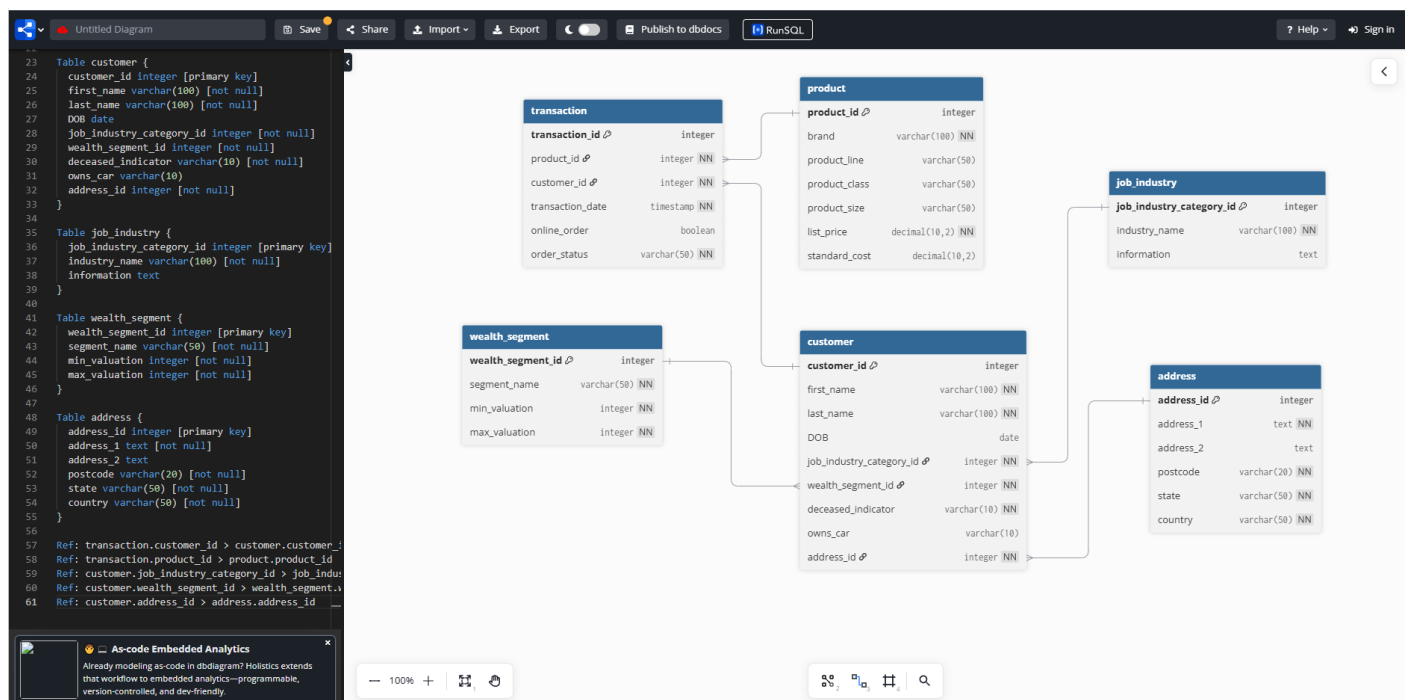
В **job\_industry**: все атрибуты зависят только от job\_industry\_category\_id

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 – ЗНФ

Далее работаем с базой данных в программе DBeaver 25.2.4.

Для начала сохраняем базу данных **customer\_and\_transaction.xlsx** в виде двух файлов: **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.

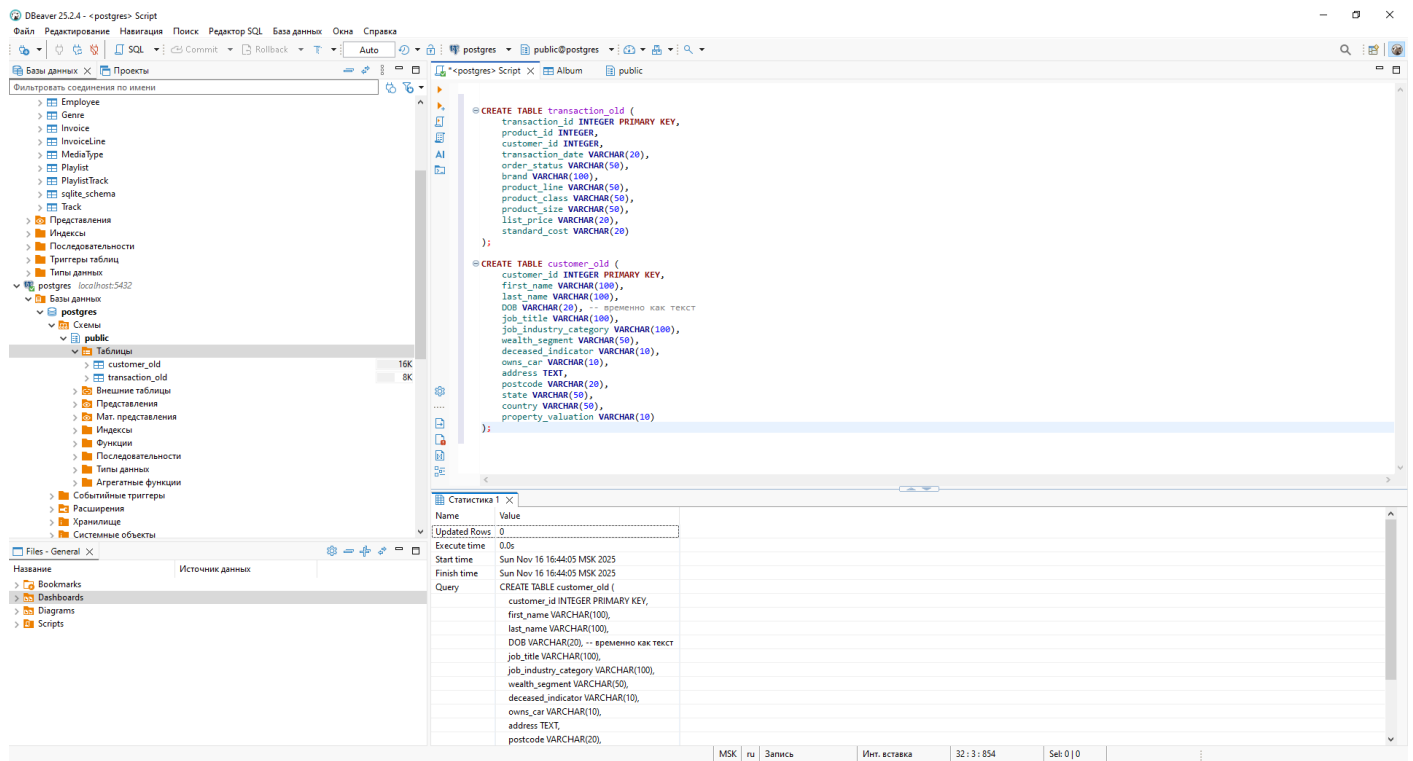


Рисунок 4 – Создание таблиц **transaction\_old** и **customer\_old**

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

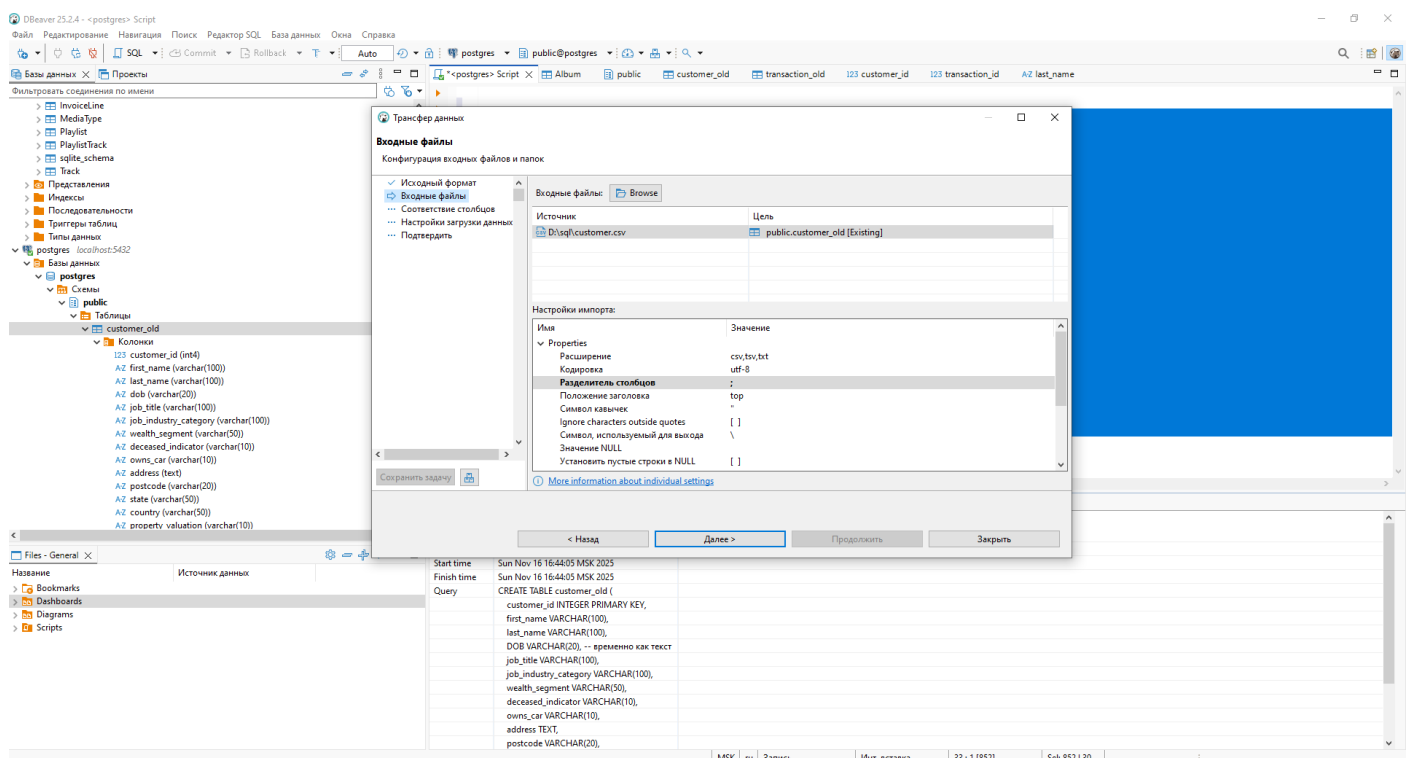


Рисунок 5 – Импорт данных в таблицу **customer\_old**

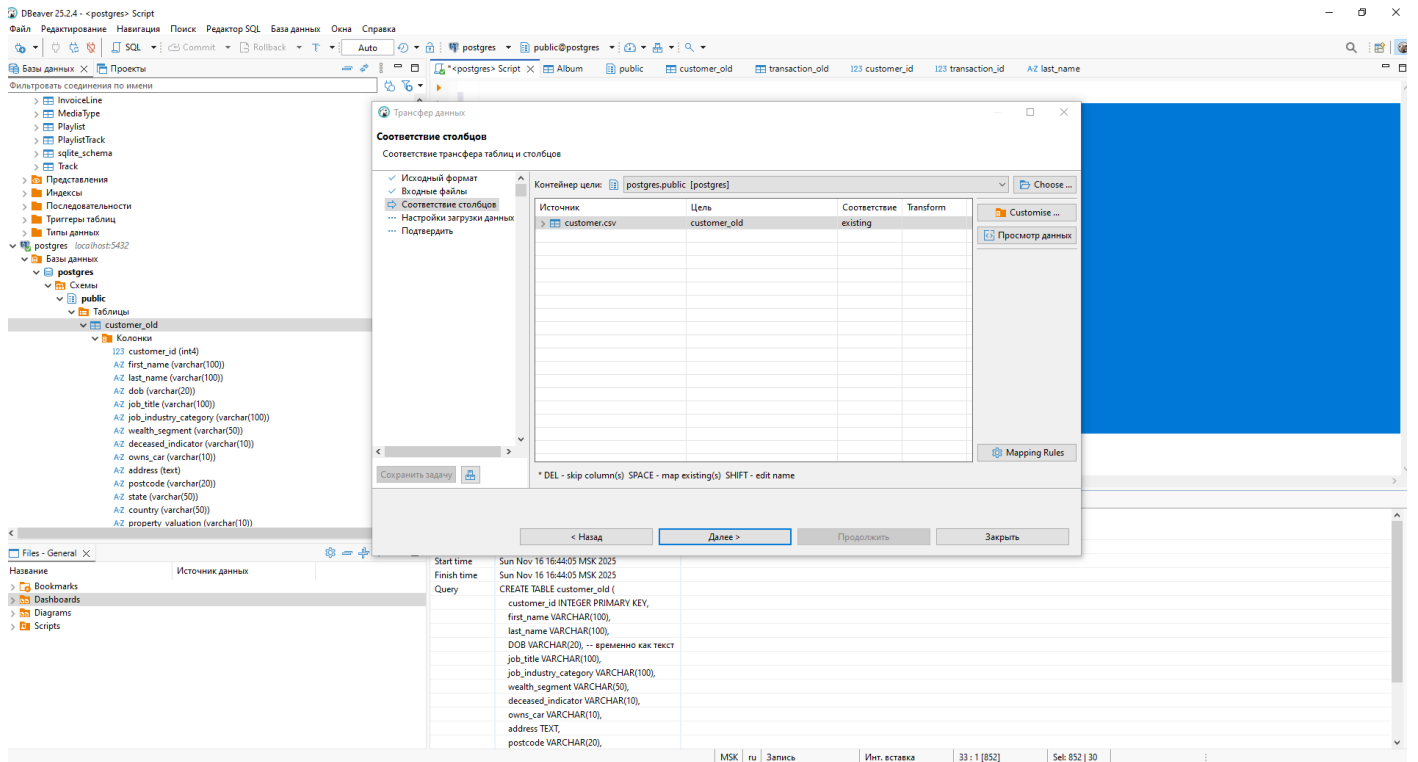


Рисунок 6 – Импорт данных в таблицу **customer\_old**

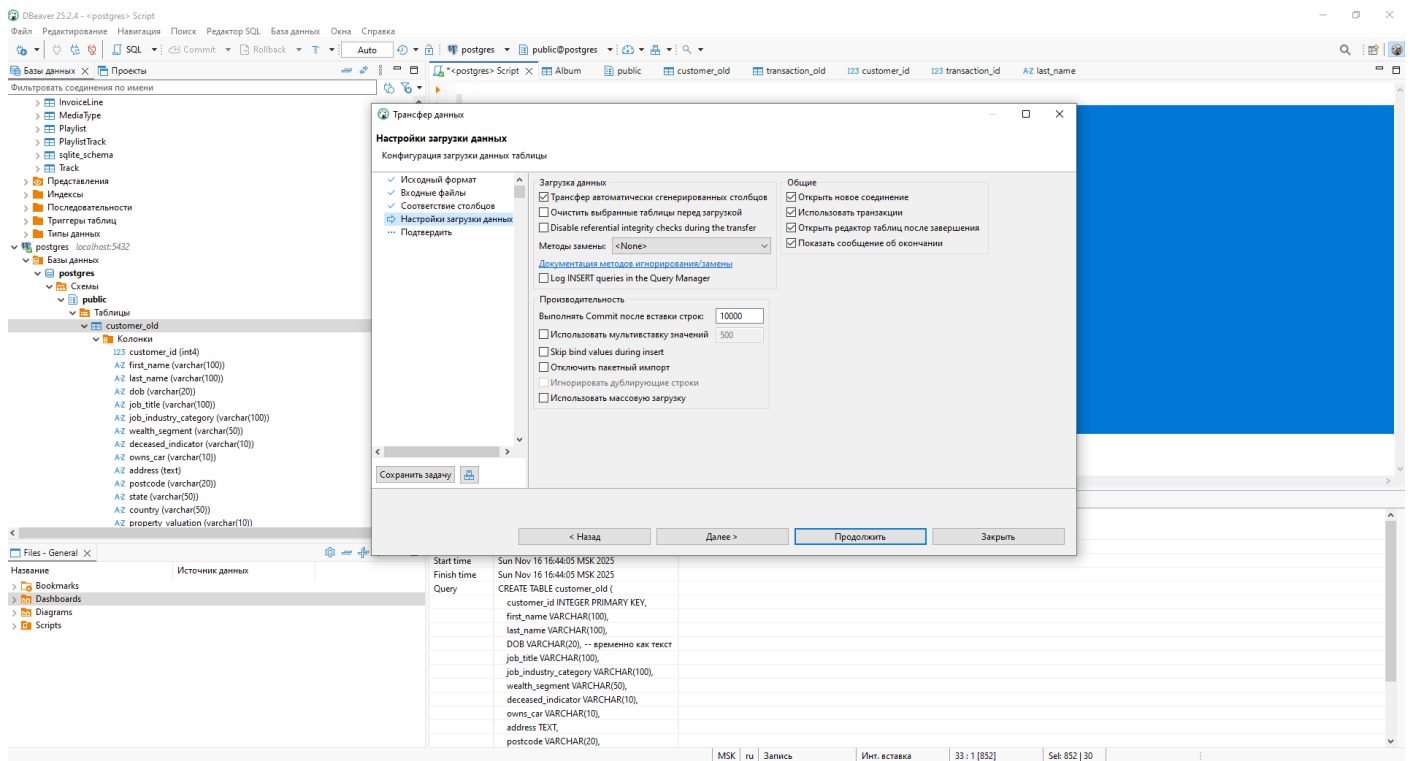


Рисунок 7 – Импорт данных в таблицу **customer\_old**



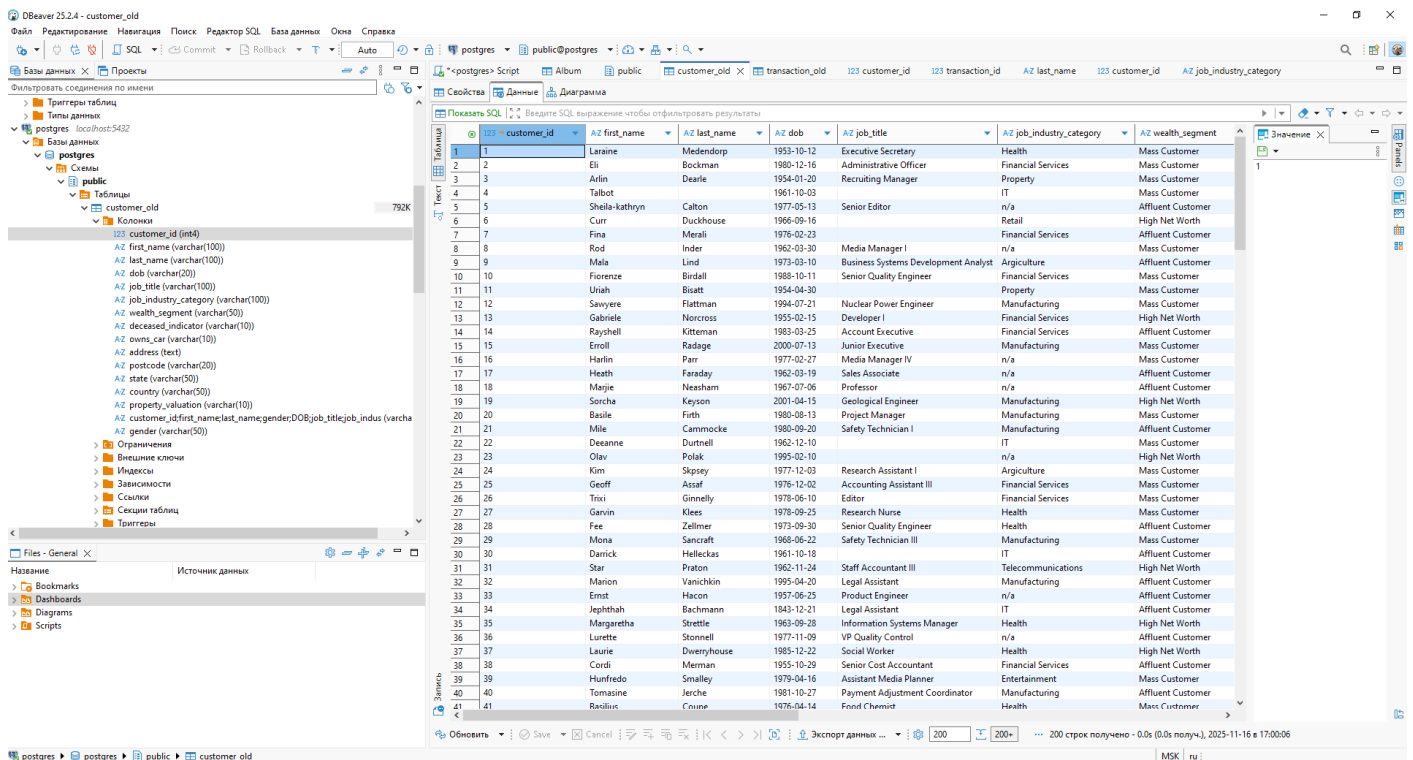


Рисунок 8 – Просмотр данных таблицы **customer\_old**

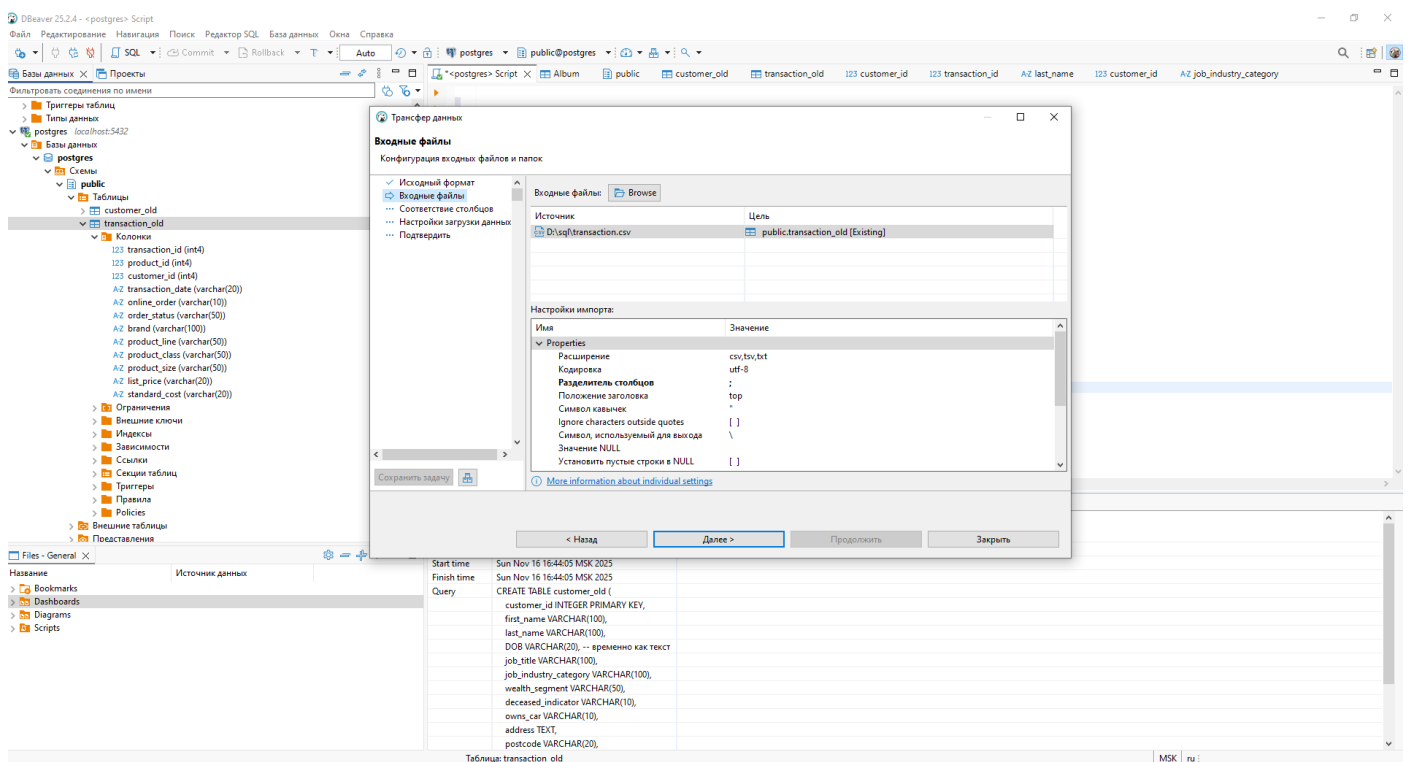


Рисунок 9 – Импорт данных в таблицу **transaction\_old**

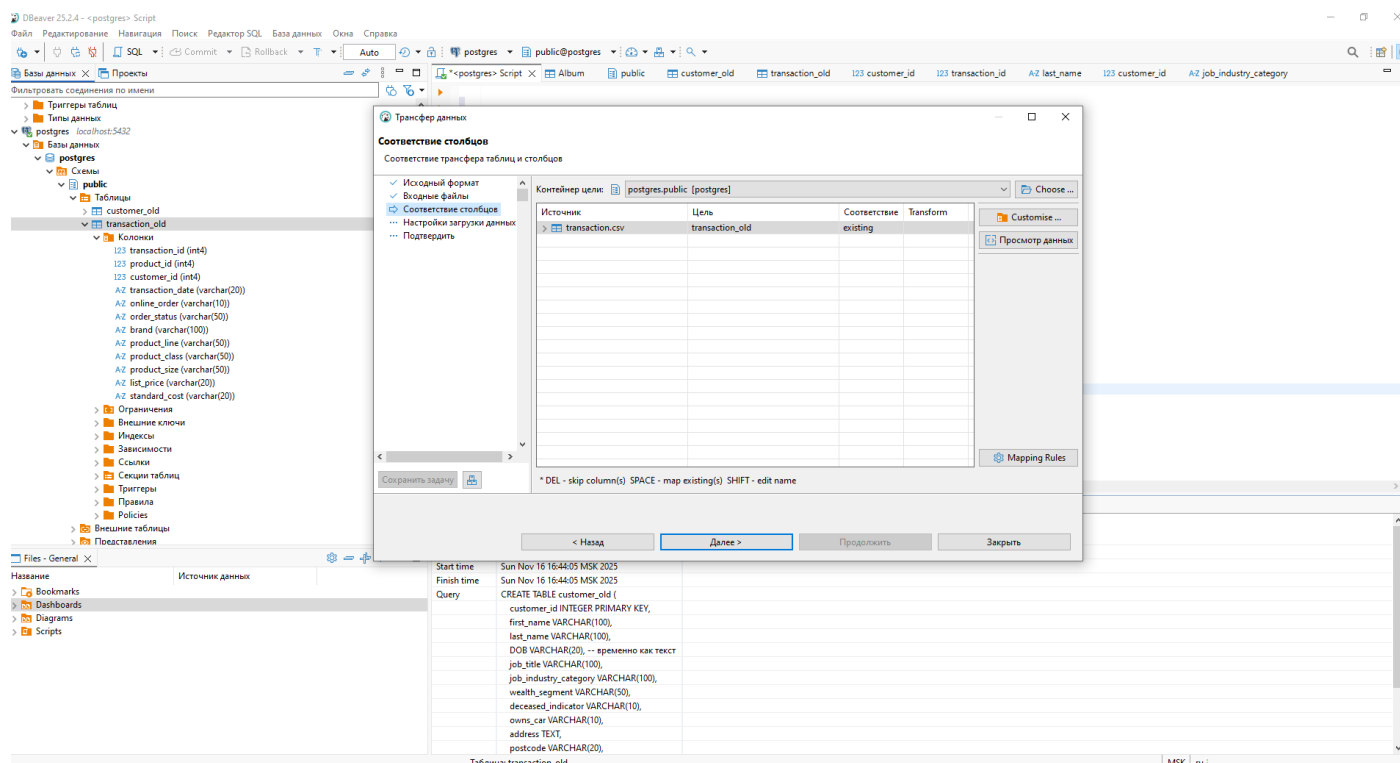


Рисунок 10 – Импорт данных в таблицу **transaction\_old**

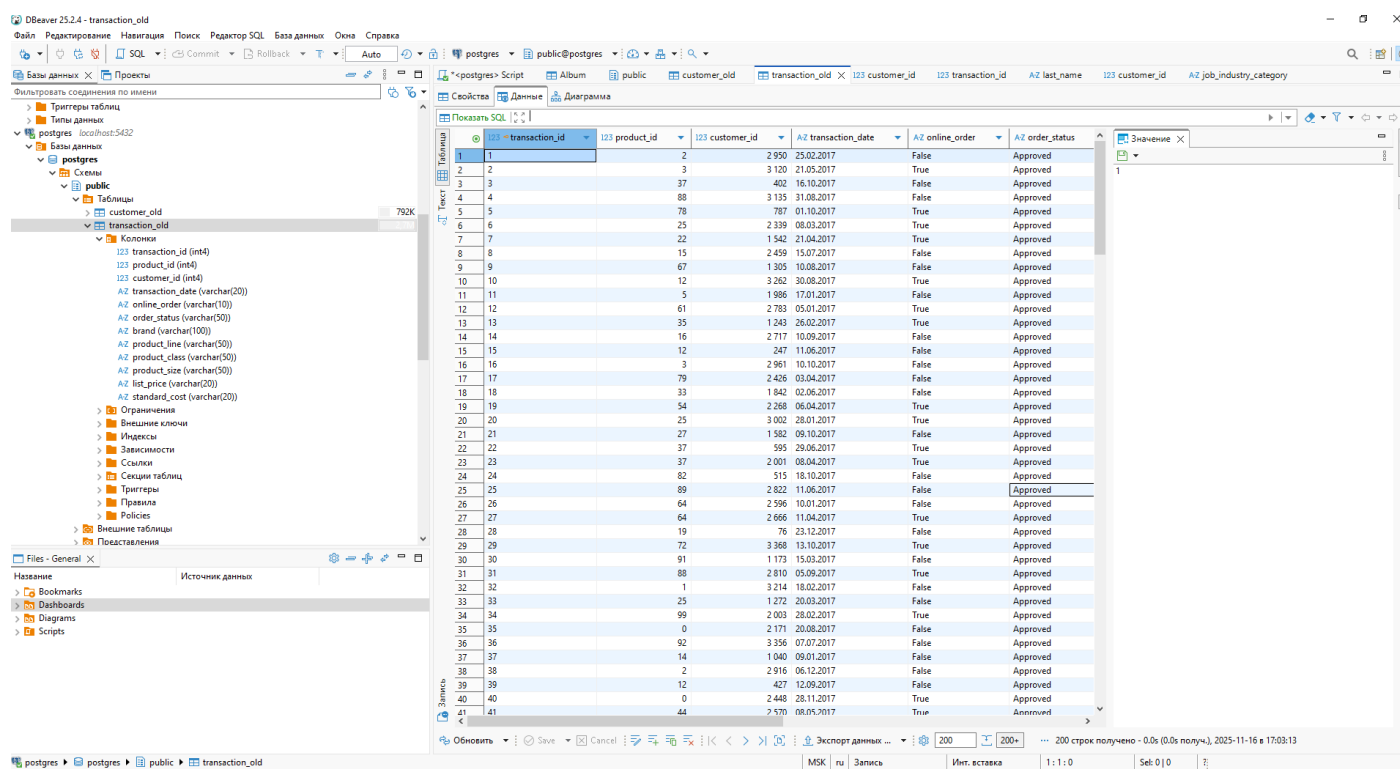


Рисунок 11 – Просмотр данных таблицы **transaction\_old**

Создаем таблицы: **job\_industry**, **wealth\_segment**, **address**, **product**, **customer**, **transaction** с колонками и типами данных, соответствующими ЗНФ.

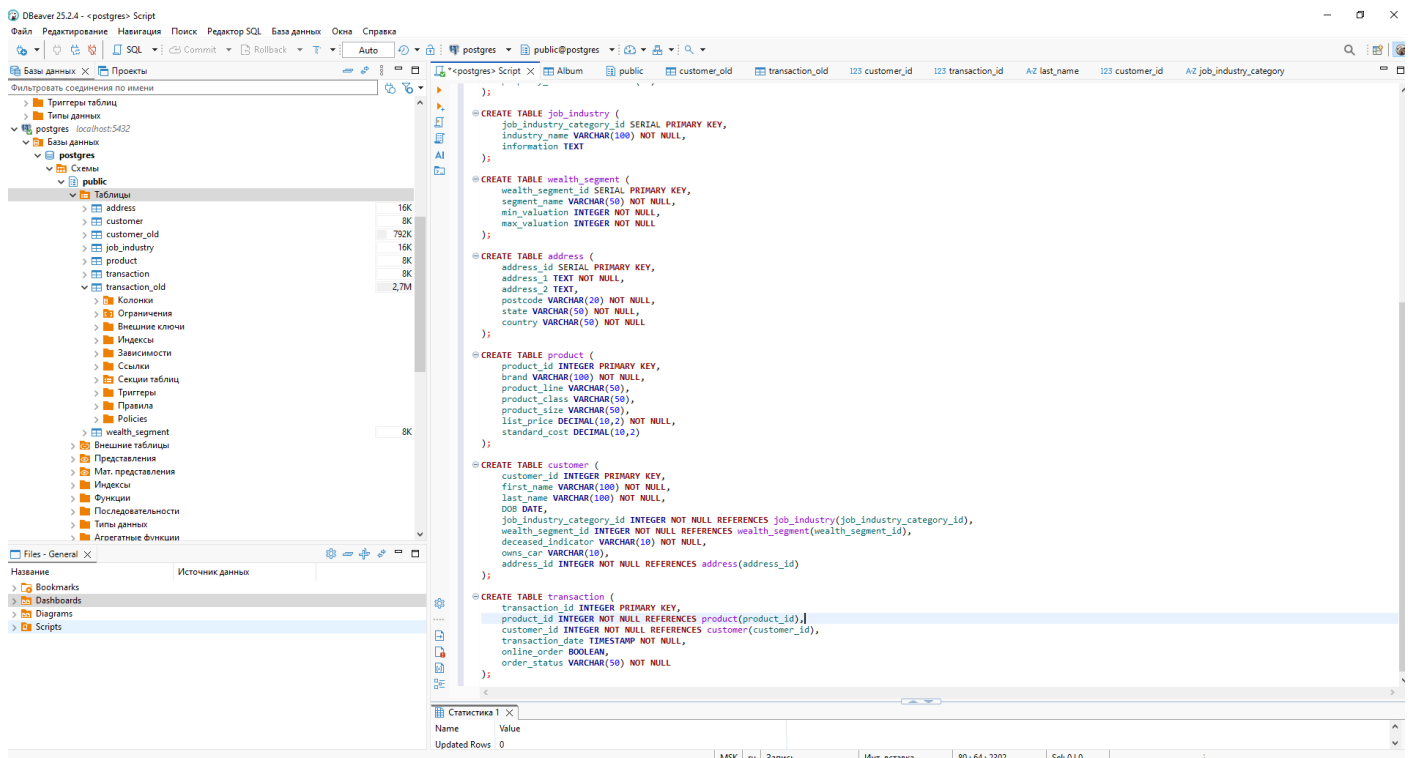


Рисунок 12 – Создание таблиц **job\_industry**, **wealth\_segment**, **address**, **product**, **customer**, **transaction**

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

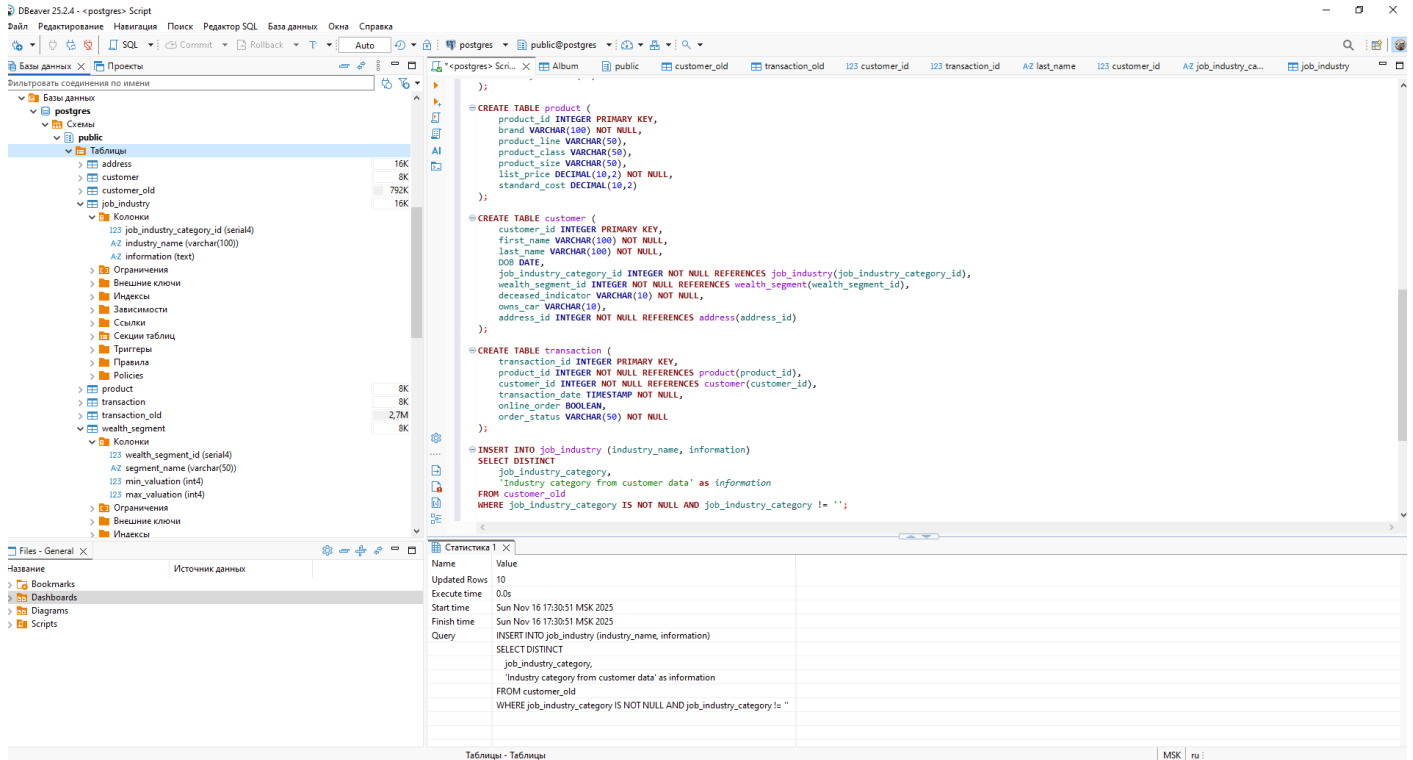


Рисунок 14 – Заполнение таблицы **job\_industry**

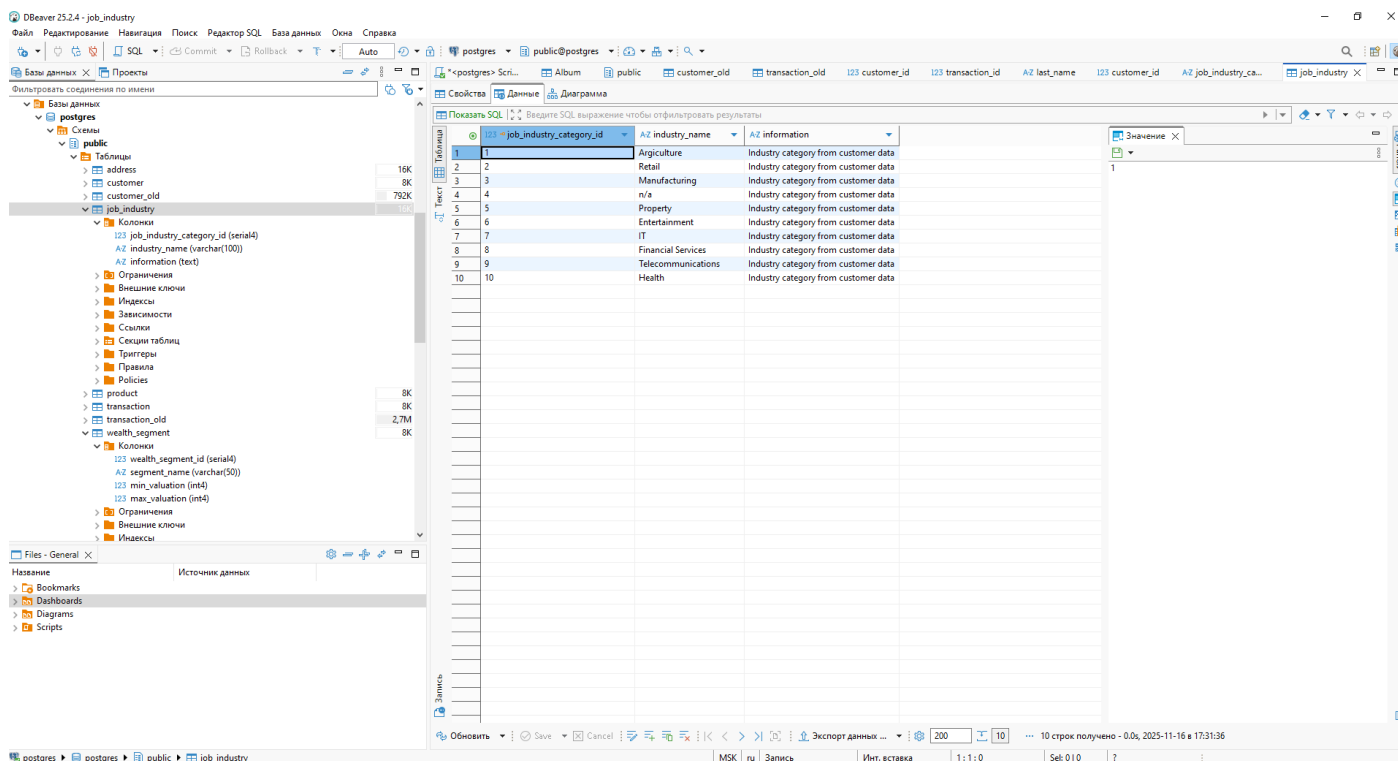


Рисунок 13 – Просмотр данных таблицы **job\_industry**

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.

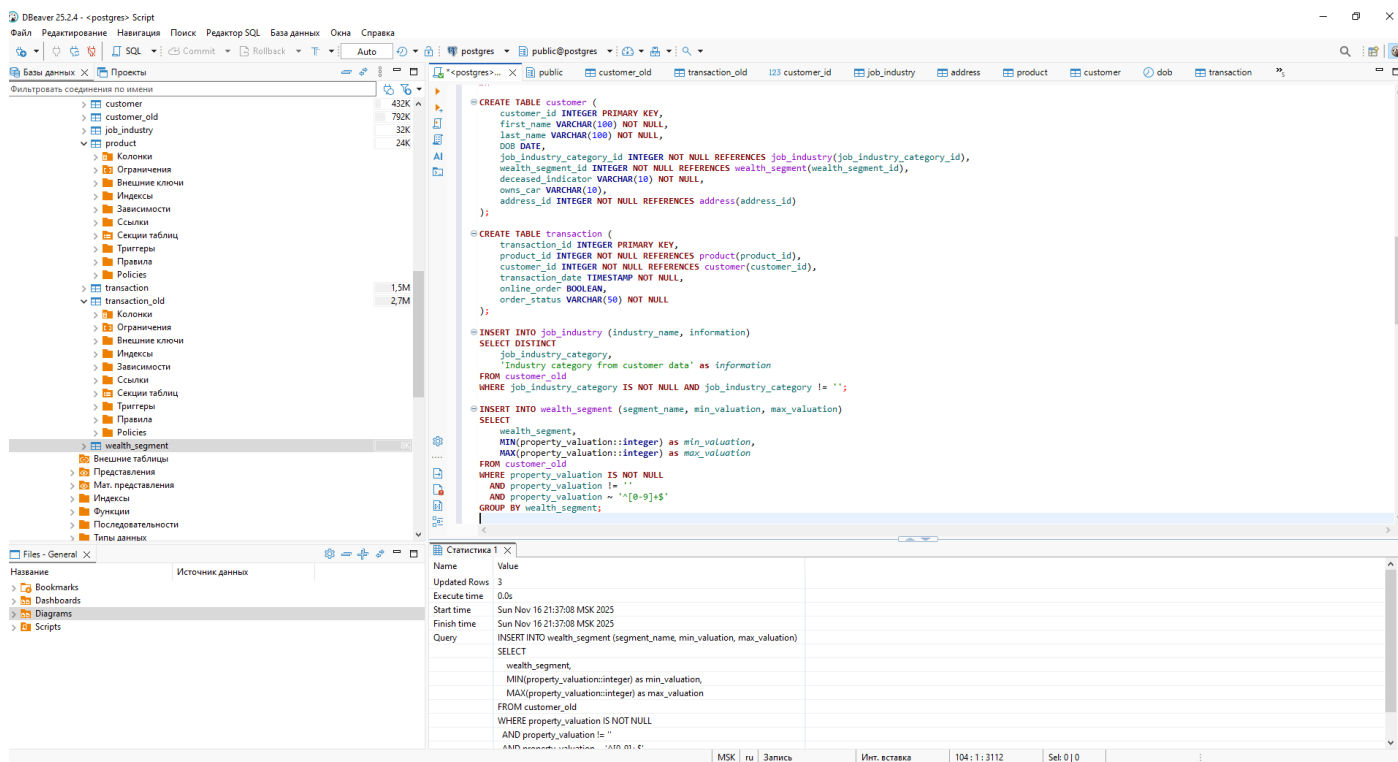


Рисунок 15 – Заполнение таблицы **wealth\_segment**

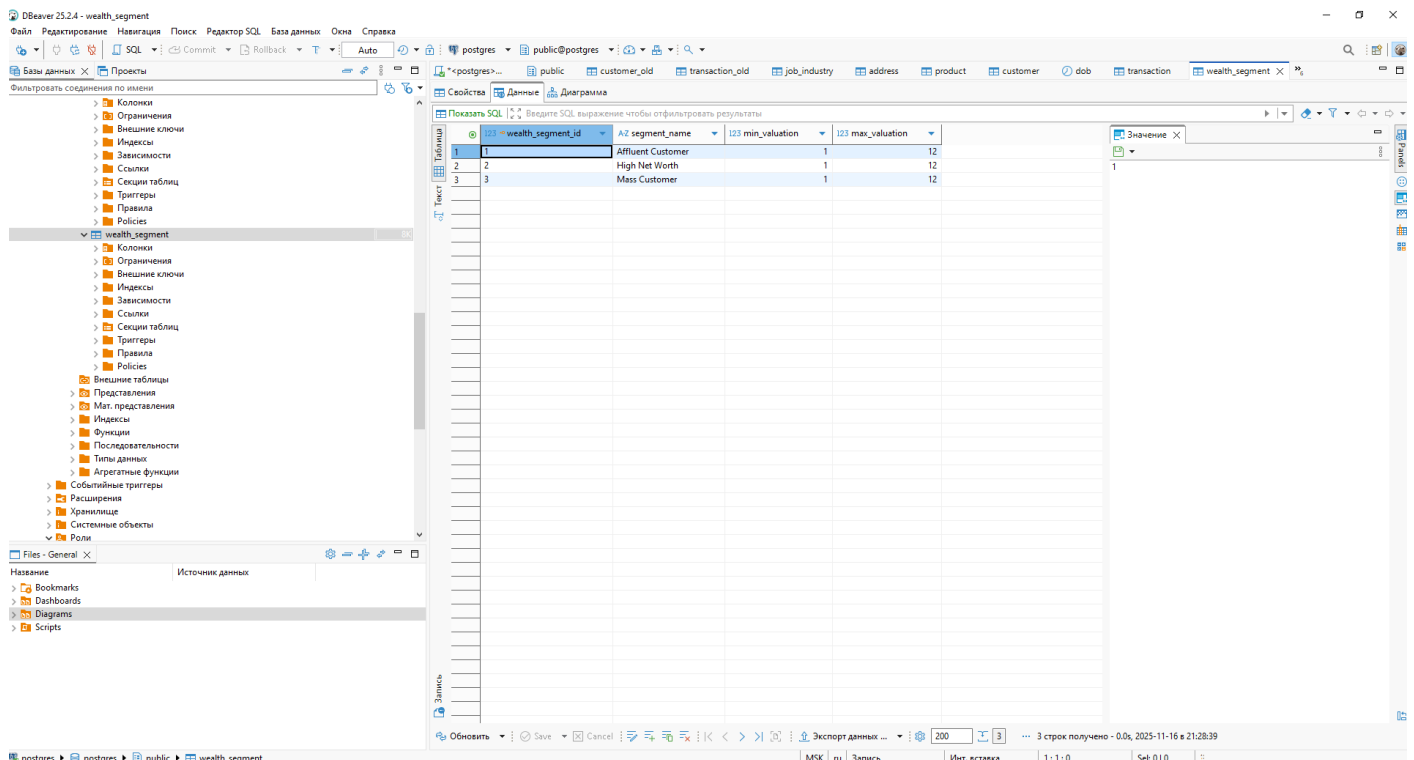


Рисунок 16 – Просмотр данных таблицы **wealth\_segment**

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.

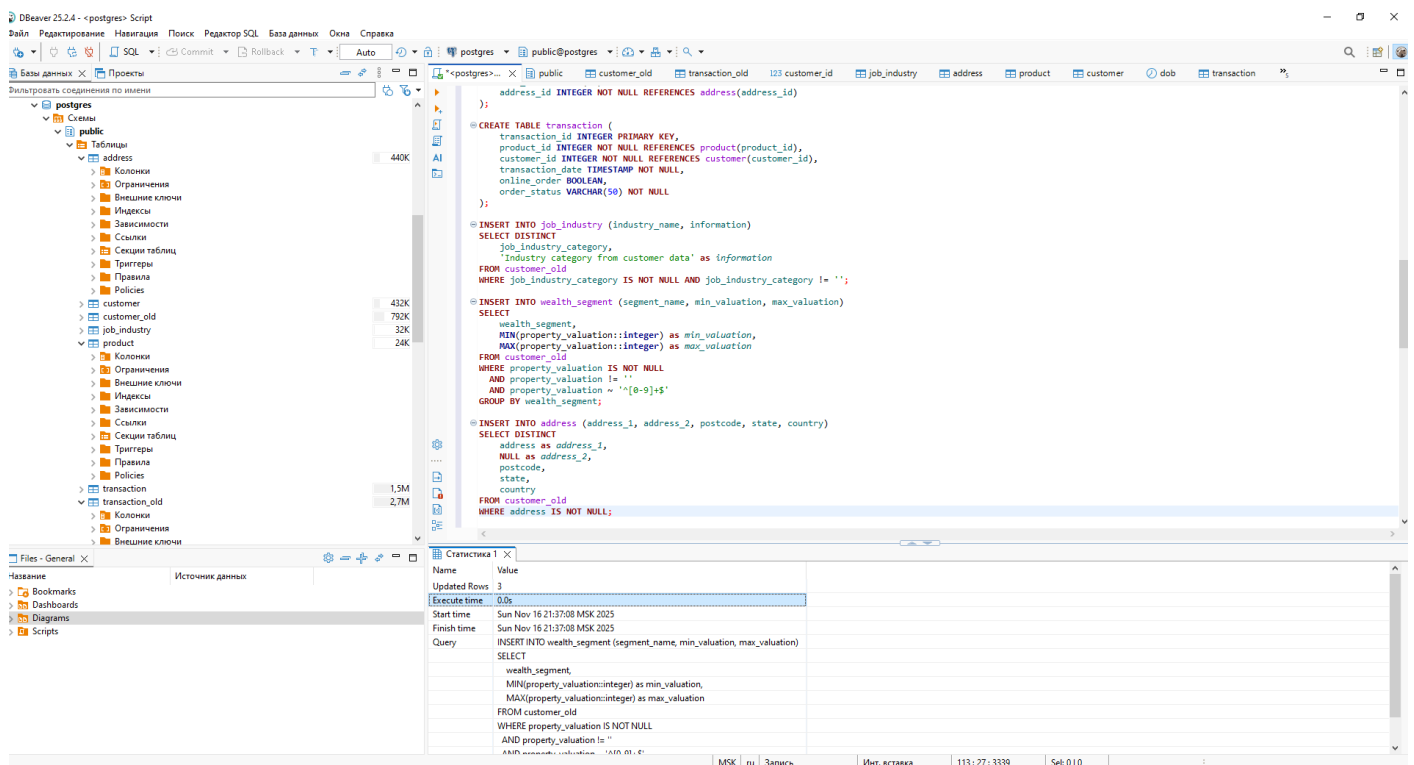


Рисунок 17 – Заполнение таблицы **address**

address_id	address_1	address_2	postcode	state	country
1	709 Rieder Terrace	[NULL]	3995	VIC	Australia
2	8462 Emmet Trail	[NULL]	4878	QLD	Australia
3	24198 Almo Crossing	[NULL]	2021	NSW	Australia
4	2388 Thackery Place	[NULL]	3130	VIC	Australia
5	7 Sumnerview Parkway	[NULL]	3073	VIC	Australia
6	31186 Hoard Junction	[NULL]	2015	NSW	Australia
7	2 Green Ridge Park	[NULL]	2478	NSW	Australia
8	1 Dayton Park	[NULL]	2767	NSW	Australia
9	434 Eggendart Circle	[NULL]	2260	NSW	Australia
10	5 Hawk Lane	[NULL]	2650	NSW	Australia
11	5015 Pawling Park	[NULL]	2525	VIC	Australia
12	5 Gale Street	[NULL]	4507	QLD	Australia
13	11 Dunning Pass	[NULL]	2219	NSW	Australia
14	9317 Mendota Parkway	[NULL]	3064	VIC	Australia
15	81856 Express Lane	[NULL]	2299	NSW	Australia
16	13 Colorado Lane	[NULL]	2099	NSW	Australia
17	54295 Dorton Hill	[NULL]	2219	NSW	Australia
18	707 Spaight Place	[NULL]	2830	NSW	Australia
19	5015 Pawling Park	[NULL]	2525	VIC	Australia
20	2390 Blackbird Point	[NULL]	2035	NSW	Australia
21	37 Darwin Circle	[NULL]	2282	NSW	Australia
22	5 Reindahl Point	[NULL]	3400	VIC	Australia
23	29 South Point	[NULL]	3111	Victoria	Australia
24	53 Dakota Court	[NULL]	2027	NSW	Australia
25	6674 Russell Center	[NULL]	2197	NSW	Australia
26	1 Arkside Avenue	[NULL]	3109	VIC	Australia
27	22 Delo Point	[NULL]	2233	NSW	Australia
28	2 Monterey Terrace	[NULL]	2120	NSW	Australia
29	7 Havey Point	[NULL]	4005	QLD	Australia
30	869 Forster Circle	[NULL]	2112	New South Wales	Australia
31	3 Bashford Plaza	[NULL]	2267	NSW	Australia
32	098 Veith Hill	[NULL]	4510	QLD	Australia
33	77729 Kim Plaza	[NULL]	2560	NSW	Australia
34	6 Golf Center	[NULL]	2042	NSW	Australia
35	087 Falkview Plaza	[NULL]	4209	QLD	Australia
36	7040 Sutteridge Lane	[NULL]	2251	NSW	Australia
37	69278 High Crossing P	[NULL]	2795	NSW	Australia
38	75717 Bartillon Road	[NULL]	2880	NSW	Australia
39	4368 Dayton Street	[NULL]	2753	NSW	Australia
40	486 Mariners Cove Pla	[NULL]	3046	VIC	Australia
41	4898 Spaight Court	[NULL]	3752	NSW	Australia

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.

```

WHERE property_valuation IS NOT NULL
AND property_valuation != ''
AND property_valuation ~ '[0-9]+[.]?'
GROUP BY wealth_segment;

-- INSERT INTO address (address_1, address_2, postcode, state, country)
SELECT DISTINCT
  address as address_1,
  NULL as address_2,
  postcode,
  state,
  country
FROM customer_old
WHERE address IS NOT NULL;

-- INSERT INTO product (product_id, brand, product_line, product_class, product_size, list_price, standard_cost)
SELECT DISTINCT ON (product_id)
  product_id,
  brand,
  product_line,
  product_class,
  product_size,
  CAST(
    CASE
      WHEN NULLIF(TRIM(list_price), '') IS NOT NULL THEN
        REPLACE(TRIM(list_price), ',', '.')
      ELSE NULL
    END AS DECIMAL(10,2)
  ) as list_price,
  CAST(
    CASE
      WHEN NULLIF(TRIM(standard_cost), '') IS NOT NULL THEN
        REPLACE(TRIM(standard_cost), ',', '.')
      ELSE NULL
    END AS DECIMAL(10,2)
  ) as standard_cost
FROM transaction_old
WHERE product_id IS NOT NULL
ORDER BY product_id, list_price DESC;

```

Name	Value
Updated Rows	3
Execute time	0.0s
Start time	Sun Nov 16 21:37:08 MSK 2025
Finish time	Sun Nov 16 21:37:08 MSK 2025
Query	INSERT INTO wealth_segment (segment_name, min_valuation, max_valuation) SELECT wealth_segment, MIN(property_valuation::integer) as min_valuation, MAX(property_valuation::integer) as max_valuation FROM customer_old WHERE property_valuation IS NOT NULL AND property_valuation != '' AND property_valuation ~ '[0-9]+[.]?';

Рисунок 19 – Заполнение таблицы **product**

product_id	brand	product_line	product_class	product_size	list_price	standard_cost
1	Giant Bicycles	Touring	medium	large	2 086,07	1 873,97
2	Giant Bicycles	Road	low	small	590,26	590,26
3	Trek Bicycles	Standard	medium	large	2 091,47	1 843,2
4	Solex	Standard	medium	medium	1 483,2	1 129,13
5	Giant Bicycles	Standard	high	medium	1 129,13	748,17
6	Solex	Standard	high	medium	748,17	1 311,44
7	Giant Bicycles	Standard	medium	small	1 703,52	1 216,14
8	Solex	Road	medium	small	1 703,52	1 945,43
9	Norco Bicycles	Standard	medium	small	1 216,14	1 775,81
10	Norco Bicycles	Standard	medium	medium	1 945,43	1 765,3
11	Trek Bicycles	Standard	medium	small	1 775,81	1 577,53
12	Giant Bicycles	Standard	medium	large	1 765,3	1 842,92
13	Solex	Standard	high	large	1 577,53	1 292,84
14	Solex	Standard	high	large	1 842,92	1 661,92
15	WearA2B	Standard	medium	medium	1 292,84	1 362,69
16	Norco Bicycles	Standard	medium	small	1 661,92	1 146,64
17	Norco Bicycles	Standard	medium	medium	1 362,69	574,64
18	Trek Bicycles	Mountain	low	medium	574,64	1 775,81
19	Trek Bicycles	Standard	medium	small	1 775,81	1 466,68
20	WearA2B	Touring	medium	medium	1 466,68	575,27
21	Solex	Standard	medium	medium	575,27	1 198,46
22	Norco Bicycles	Standard	medium	medium	1 198,46	1 777,8
23	Solex	Road	medium	large	1 777,8	2 005,66
24	OHM Cycles	Standard	high	medium	2 005,66	1 992,93
25	WearA2B	Standard	medium	medium	1 992,93	1 057,51
26	Trek Bicycles	Standard	low	medium	1 057,51	1 703,52
27	Solex	Road	medium	small	1 703,52	1 065,03
28	WearA2B	Standard	medium	medium	1 065,03	1 227,34
29	OHM Cycles	Standard	medium	medium	1 227,34	752,64
30	WearA2B	Standard	medium	medium	752,64	1 179
31	Giant Bicycles	Standard	high	medium	1 179	1 810
32	Giant Bicycles	Standard	high	medium	1 810	1 231,15
33	OHM Cycles	Road	medium	small	1 231,15	1 403,5
34	WearA2B	Standard	medium	medium	1 403,5	1 289,85
35	Giant Bicycles	Standard	medium	medium	1 289,85	1 793,43
36	Solex	Standard	low	medium	1 793,43	2 091,47
37	OHM Cycles	Standard	low	medium	2 091,47	1 812,75
38	Trek Bicycles	Standard	medium	large	1 812,75	1 894,14
39	Giant Bicycles	Standard	medium	large	1 894,14	
40	Trek Bicycles	Road	medium	large		

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.

```

REPLACE(TRIM(list_price), ',', '.')
ELSE NULL
END AS DECIMAL(10,2)
) as list_price,
CASE
WHEN NULLIF(TRIM(standard_cost), '') IS NOT NULL THEN
REPLACE(TRIM(standard_cost), ',', '.')
ELSE NULL
END AS DECIMAL(10,2)
) as standard_cost
FROM transaction_old
WHERE product_id IS NOT NULL
ORDER BY product_id, list_price DESC;

INSERT INTO customer (
customer_id, first_name, last_name, DOB,
job_industry_category_id, wealth_segment_id,
deceased_indicator, owns_car, address_id
)
SELECT
c.customer_id,
c.first_name,
c.last_name,
CASE
WHEN c.DOB ~ '^([0-9]{4})-([0-9]{2})-([0-9]{2})$' THEN c.DOB::DATE
ELSE NULL
END as DOB,
j.job_industry_category_id,
ws.wealth_segment_id,
c.deceased_indicator,
c.owns_car,
a.address_id
FROM customer_old c
LEFT JOIN job_industry j ON c.job_industry_category = j.job_industry_name
LEFT JOIN wealth_segment ws ON c.wealth_segment = ws.wealth_segment_name
LEFT JOIN address a ON c.address = a.address_1 AND c.postcode = a.postcode;

```

Рисунок 21 – Заполнение таблицы **customer**







transaction_id	product_id	customer_id	transaction_date	online_order	A2_order_status
1	2	2950	[NULL]	[ ]	Approved
2	3	3120	[NULL]	[v]	Approved
3	37	402	[NULL]	[ ]	Approved
4	88	3135	[NULL]	[ ]	Approved
5	78	787	[NULL]	[v]	Approved
6	25	2339	[NULL]	[v]	Approved
7	22	1542	[NULL]	[v]	Approved
8	15	2459	[NULL]	[ ]	Approved
9	67	1305	[NULL]	[ ]	Approved
10	12	3262	[NULL]	[v]	Approved
11	5	1986	[NULL]	[ ]	Approved
12	61	2783	[NULL]	[v]	Approved
13	35	1243	[NULL]	[v]	Approved
14	16	2717	[NULL]	[ ]	Approved
15	12	247	[NULL]	[ ]	Approved
16	3	2961	[NULL]	[ ]	Approved
17	79	2426	[NULL]	[ ]	Approved
18	33	1842	[NULL]	[ ]	Approved
19	54	2268	[NULL]	[v]	Approved
20	25	3002	[NULL]	[v]	Approved
21	27	1582	[NULL]	[ ]	Approved
22	37	595	[NULL]	[v]	Approved
23	37	2001	[NULL]	[v]	Approved
24	82	515	[NULL]	[ ]	Approved
25	89	2822	[NULL]	[ ]	Approved
26	64	2596	[NULL]	[ ]	Approved
27	64	2666	[NULL]	[v]	Approved
28	19	76	[NULL]	[ ]	Approved
29	72	3368	[NULL]	[v]	Approved
30	91	1173	[NULL]	[ ]	Approved
31	88	2810	[NULL]	[v]	Approved
32	1	3214	[NULL]	[ ]	Approved
33	25	1272	[NULL]	[ ]	Approved
34	99	2003	[NULL]	[v]	Approved
35	0	2171	[NULL]	[ ]	Approved
36	92	3356	[NULL]	[ ]	Approved
37	14	1040	[NULL]	[ ]	Approved
38	2	2916	[NULL]	[ ]	Approved
39	12	427	[NULL]	[ ]	Approved
40	0	2448	[NULL]	[v]	Approved
41	44	2570	[NULL]	[v]	Disapproved

Рисунок 24 – Просмотр данных таблицы **transaction**

Преобразованную до ЗНФ базу данных в формате .csv экспортированную из таблиц: **job\_industry**, **wealth\_segment**, **address**, **product**, **customer**, **transaction** можно найти в папке “Экспортируемые итоговые csv”.