

Министерство науки и высшего образования Российской Федерации

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

Факультет инфокоммуникационных технологий

ЛАБОРАТОРНАЯ РАБОТА №2
«Запросы на выборку и
модификацию данных,
представления и индексы в
PostgreSQL»

Выполнила:
студент : Аль-Мошки Исмаил
Абдулвахаб
группа: К32401

Проверили:
Говорова Марина Михайловна

Санкт-Петербург
2023

Цель работы: овладеть практическими навыками создания представлений и запросов на выборку данных к базе данных PostgreSQL, использования подзапросов при модификации данных и индексов.

Оборудование: компьютерный класс.

Программное обеспечение: СУБД PostgreSQL, pgadmin 4.

Практическое задание:

1. Создать запросы и представления на выборку данных к базе данных PostgreSQL (согласно индивидуальному заданию, часть 2 и 3).

Вариант 12. БД «Прокат автомобилей»

Описание предметной области: Компания предоставляет прокат автомобилей. В пункт проката обращаются клиенты, данные о которых регистрируют в базе. Цена проката зависит от марки автомобиля, технических характеристик и года выпуска.

Для проката авто с клиентом заключается договор, в котором фиксируется период проката, вид страховки, стоимость страховки, залоговая стоимость. Залоговая стоимость возвращается полностью или частично клиенту, в зависимости от страховки, аварий и штрафов. Если залоговая стоимость уже возвращена клиенту, но на авто в компанию пришел штраф, то он оплачивается компанией, а не клиентом. При передаче авто клиенту составляется акт о передаче автомобиля клиенту. При возвращении автомобиля также составляется акт о передаче авто компании.

Если клиент не вернул автомобиль в срок и не оформил продление, ему назначается штраф за каждый час просрочки.

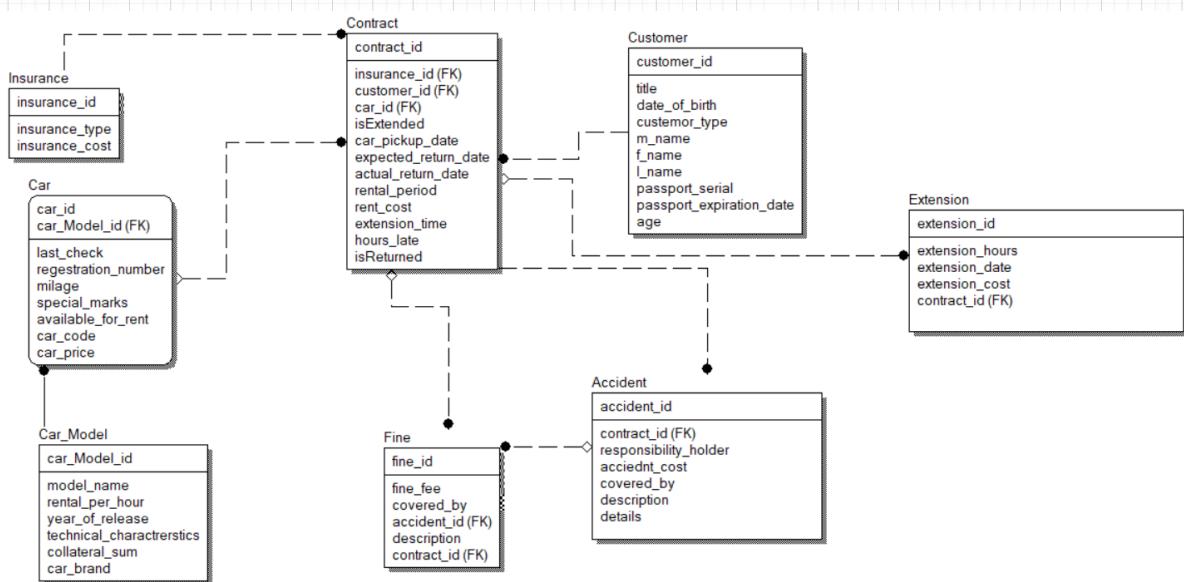
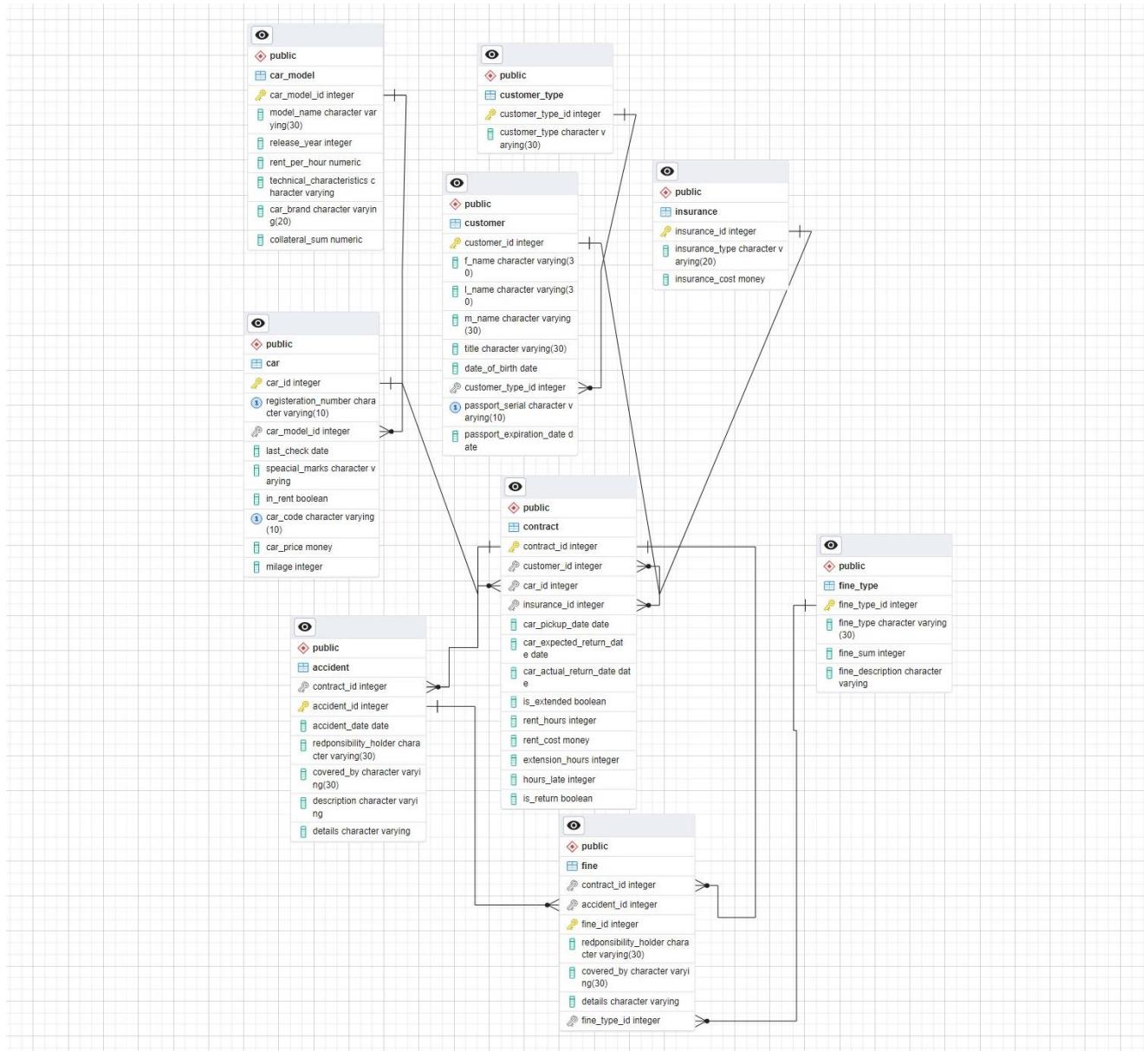
Постоянным клиентам предоставляются скидки.

В системе необходимо хранить историю штрафов и аварий автомобилей.

Цены на прокат автомобилей могут меняться.

БД должна содержать следующий минимальный набор сведений: ФИО. Паспортные данные. Код должности. Наименование должности. Оклад. Обязанности. Код марки. Наименование. Технические характеристики. Описание. Код автомобиля. Регистрационный номер. Номер кузова. Номер двигателя. Год выпуска. Пробег. Цена автомобиля. Цена проката. Дата последнего ТО. Специальные отметки. Отметка о возврате. Код клиента. ФИО. Адрес. Телефон. Паспортные данные. Дата и время выдачи автомобиля. На сколько часов. Дата и время возврата автомобиля. Данные о нарушениях. Данные об авариях. Дата продления. Часов продления.

Схема базы данных:



Задание 2. Создать запросы:

- Какой автомобиль находился в прокате максимальное количество часов?
- SELECT car_id from contract JOIN car USING(car_id) GROUP BY car_id HAVING sum (rent_hours) =

```
(SELECT max(sum) FROM
(SELECT sum(rent_hours) FROM contract JOIN car using(car_id) GROUP BY
car_id) as temp1)
```

The screenshot shows the pgAdmin 4 interface. At the top, there's a navigation bar with tabs for Properties, Dashboard, SQL, Statistics, Dependencies, Dependents, Processes, and two database connections: 'Car Rental/postgres@PostgreSQL 15' and 'Untitled*'. Below the navigation bar is a toolbar with various icons for file operations, search, and database management.

The main area has three tabs: Messages, Query, and Notifications. The 'Query' tab is selected, displaying the following SQL code:

```
1 SELECT car_id from contract JOIN car USING(car_id) GROUP BY car_id
2 HAVING sum (rent_hours) =
3 (SELECT max(sum) FROM
4 (SELECT sum(rent_hours) FROM contract JOIN car using(car_id) GROUP BY car_id) as temp1)
5
```

Below the query editor is a 'Data Output' tab, which is currently active. It shows a table with one row of data:

car_id	integer
1	4

- Автомобили какой марки чаще всего брались в прокат?
- SELECT car_id from contract JOIN car USING(car_id) GROUP BY car_id
HAVING count(rent_hours) =

```
(SELECT max(count) FROM (SELECT sum(rent_hours) FROM contract JOIN car using(car_id)
GROUP BY car_id) as temp1)
```

The screenshot shows the pgAdmin 4 interface. At the top, there's a navigation bar with links like Properties, Dashboard, SQL, Statistics, Dependencies, Dependents, Processes, and Car Rental/pos... (with Untitled* and Car Rental/pos...). Below the navigation bar is a toolbar with various icons for file operations, search, and database management.

The main area has three tabs: Messages, Query, and Notifications. The Query tab is active, displaying the following SQL code:

```
1 SELECT car_id from contract JOIN car USING(car_id) GROUP BY car_id
2 HAVING count(rent_hours) =
3 (SELECT max(hr) FROM
4 (SELECT count(rent_hours) as hr FROM contract JOIN car using(car_id) GROUP BY car_id) as
```

Below the Query tab is a Data Output tab, which is also active. It shows a table with one column named 'car_id' and two rows of data:

car_id
integer
1
2

- Определить убытки от простоя автомобилей за вчерашний день.
- Вывести данные автомобиля, имеющего максимальный пробег.

```
SELECT * FROM car GROUP BY car_id HAVING milage = (SELECT max(milage) FROM car)
```

The screenshot shows the pgAdmin 4 interface. On the left, the database browser pane lists various database objects: FTS Dictionaries, FTS Parsers, FTS Templates, Foreign Tables, Functions, Materialized Views, Operators, Procedures, Sequences, Tables (9), accident, car, and car_model. The car_model table is selected, and its columns (car_model_id, model_name, release_year, rent_per_hour, technical_characteristic, car_brand, collateral_sum) are listed. Below the table structure, there are sections for Constraints, Indexes, RLS Policies, Rules, and Triggers.

In the main pane, a query is being run:

```

SELECT car_model.car_brand , car_model.model_name, count(*) cnt FROM car_model
JOIN car ON car.car_model_id = car_model.car_model_id
JOIN contract ON contract.car_id = car.car_id
GROUP BY car_model.car_brand , car_model.model_name

```

The results of the query are displayed in a table:

car_brand	model_name	cnt
Jeep	JEEP CHEROKI	4
Mazda	TOYOTA	3

Total rows: 2 of 2 Query complete 00:00:00.085 Ln 4, Col 52

- Какой автомобиль суммарно находился в прокате дольше всех.

SELECT * FROM

(SELECT car_id, sum(rent_hours) AS hours_sum FROM contract GROUP BY car_id)a
GROUP BY a.car_id,a.hours_sum HAVING a.hours_sum = max(a.hours_sum)

The screenshot shows the pgAdmin 4 interface with a different query in the editor:

```

1 SELECT * FROM
2 (SELECT car_id, sum(rent_hours) AS hours_sum FROM contract GROUP BY car_id )a
3 GROUP BY a.car_id,a.hours_sum HAVING a.hours_sum = max(a.hours_sum)
4

```

The data output pane shows the results of the query:

car_id	hours_sum
1	3
	150

- Определить, каким количеством автомобилей каждой марки и модели владеет компания.

```
SELECT car_brand ,model_name, count(*) FROM car JOIN car_model ON car.car_model_id = car_model.car_model_id GROUP BY car_model.car_brand, car_model.model_name
```

The screenshot shows the pgAdmin 4 interface. On the left, the object browser displays a tree structure of database objects, including tables like accident, car, and car_model, and their columns. The car_model table is currently selected, showing its seven columns: car_model_id, model_name, release_year, rent_per_hour, technical_characterist, car_brand, and collateral_sum. In the center, the SQL tab contains the query:

```
SELECT car_brand ,model_name, count(*) FROM car
JOIN car_model ON car.car_model_id = car_model.car_model_id
GROUP BY car_model.car_brand, car_model.model_name
```

Below the query, the results grid shows the following data:

	car_brand	model_name	count
1	Jeep	Hyundai	1
2	Jeep	JEEP CHEROKI	2
3	Mazda	TOYOTA	1
4	Mazda	Mazda car	3
5	Mazda	Mitsubishi	2

Total rows: 5 of 5 Query complete 00:00:00.133 Ln 3, Col 1

- Определить средний “возраст” автомобилей компании.

```
SELECT AVG(a.age) FROM (SELECT *, date_part('year', CURRENT_DATE)- release_year AS age FROM car_model)
```

The screenshot shows the pgAdmin 4 interface. The object browser on the left shows the car_model table selected, with its columns listed. The SQL tab contains the query:

```
SELECT AVG(a.age) FROM
(SELECT *, date_part('year', CURRENT_DATE)- release_year AS age
FROM car_model
JOIN car
ON car.car_model_id = car_model.car_model_id) a
```

Below the query, the results grid shows the following data:

	avg
1	6.666666666666667

Total rows: 1 of 1 Query complete 00:00:00.131 Ln 5, Col 3

Задание 3. Создать представление:

- Какой автомобиль ни разу не был в прокате?

```
CREATE VIEW unused_cars AS
SELECT * FROM car WHERE car_id NOT IN (SELECT car_id FROM contract)
```

The screenshot shows the pgAdmin 4 interface. On the left, the Object Browser displays the database structure under 'Car Rental'. In the center, the 'Query' tab contains the SQL code for creating the 'unused_cars' view. Below the query, the 'Data Output' tab shows a table with 9 rows of data, which are the results of the query. The table has columns: car_id, registration_number, car_model_id, last_check_date, speacial_marks, in_rent, car_code, and car_type. The data is as follows:

	car_id	registration_number	car_model_id	last_check_date	speacial_marks	in_rent	car_code	car_type
1	1	12345		2021-01-01	[null]	false	run123	1
2	2	12245		2020-01-01	[null]	false	rat123	1
3	3	12445		2020-01-01	[null]	false	run432	1
4	4	12235		2020-01-01	[null]	false	rat654	1
5	5	16656		2020-01-01	[null]	false	run875	1
6	6	12647		2020-01-01	[null]	false	rat154	1
7	7	24453		2020-01-01	[null]	false	run134	1
8	8	23445		2020-01-01	[null]	false	rat243	1
9	9	435246		2020-01-05	[null]	false	run321	1

Total rows: 9 of 9 Query complete 00:00:00.093 Ln 2, Col 68

- Вывести данные клиентов, не вернувших автомобиль вовремя.

```
CREATE VIEW late_returners AS SELECT
DISTINCT customer.customer_id , customer.f_name, customer.l_name, m_name
FROM customer JOIN contract ON contract.customer_id = customer.customer_id
WHERE (contract.hours_late IS NOT NULL AND contract.hours_late > 0)
```

The screenshot shows the pgAdmin 4 interface. At the top, there's a toolbar with various icons for database management. Below the toolbar, the title bar displays "Car Rental/postgres@PostgreSQL 15". The main area has tabs for "Messages", "Query" (which is selected), "Notifications", and "Query History". The "Query" tab contains the following SQL code:

```

CREATE VIEW late_returners AS SELECT
  DISTINCT customer.customer_id , customer.f_name, customer.l_name, m_name
  FROM customer JOIN contract ON contract.customer_id = customer.customer_id
  WHERE (contract.hours_late IS NOT NULL AND contract.hours_late > 0)

SELECT * FROM late_returners

```

Below the query editor is a "Data Output" viewer. It has its own toolbar and displays a table with three rows of data:

	customer_id	f_name	l_name	m_name
1	4	MOhameed	Almoski	ABDO
2	1	Kozman	Hesham	Nasher
3	3	ABDULwahab	ESMAIL	Almoshki

At the bottom of the pgAdmin window, there are status messages: "Total rows: 3 of 3 Query complete 00:00:00.092" and "Ln 7. Col 1".

2. Составить 3 запроса на модификацию данных (INSERT, UPDATE, DELETE) с использованием подзапросов.

1-INSERT

Update car SET milage = ((SELECT max(milage) FROM car) +1000) WHERE car_id = 1

Properties Dashboard SQL Statistics Dependencies Dependents Processes Car Rental/pos... Untitled* Car Rental/pos... < > Re

Car Rental/postgres@PostgreSQL 15 No limit

Messages Query Notifications Query History

```

1 SELECT * FROM car
2
3 Update car SET milage = (SELECT max(milage) FROM car ) +1000 WHERE car_model = 3
4

```

Data Output Explain ×

	car_model_id	last_check_date	speacial_marks	in_rent	car_code	car_price	milage
	integer	date	character varying	boolean	character varying (10)	money	integer
1		1 2021-01-01	[null]	false	run123	12,000.00 .₪ .J	6000
2		2 2020-01-01	[null]	false	rat123	11,000.00 .₪ .J	6000
3		4 2020-01-01	[null]	false	run432	12,000.00 .₪ .J	6000
4		1 2020-01-01	[null]	false	rat654	11,500.00 .₪ .J	6000
5		5 2020-01-01	[null]	false	run875	13,000.00 .₪ .J	6000
6		2 2020-01-01	[null]	false	rat154	12,000.00 .₪ .J	6000
7		2 2020-01-01	[null]	false	run134	11,000.00 .₪ .J	6000
8		3 2020-01-01	[null]	false	rat243	16,000.00 .₪ .J	6000

Properties Dashboard SQL Statistics Dependencies Dependents Processes Car Rental/pos... Untitled* Car Rental/pos... < > Re

Car Rental/postgres@PostgreSQL 15 No limit

Messages Query Notifications Query History

```

1 Update car SET milage = ((SELECT max(milage) FROM car ) +1000) WHERE car_id = 1
2
3 SELECT * FROM car
4

```

Data Output Explain ×

	car_model_id	last_check_date	speacial_marks	in_rent	car_code	car_price	milage
	integer	date	character varying	boolean	character varying (10)	money	integer
1		2 2020-01-01	[null]	false	rat123	11,000.00 .₪ .J	6000
2		4 2020-01-01	[null]	false	run432	12,000.00 .₪ .J	6000
3		1 2020-01-01	[null]	false	rat654	11,500.00 .₪ .J	6000
4		5 2020-01-01	[null]	false	run875	13,000.00 .₪ .J	6000
5		2 2020-01-01	[null]	false	rat154	12,000.00 .₪ .J	6000
6		2 2020-01-01	[null]	false	run134	11,000.00 .₪ .J	6000
7		3 2020-01-01	[null]	false	rat243	16,000.00 .₪ .J	6000
8		1 2021-01-01	[null]	false	run123	12,000.00 .₪ .J	7000

2-DELETE

```
DELETE FROM contract
WHERE contract.car_id IN
(SELECT DISTINCT car.car_id FROM car JOIN car_model
ON car.car_model_id = car_model.car_model_id
WHERE car_model.release_year= (SELECT max(release_year)FROM car_model) )
```



```
1 SELECT * FROM car JOIN car_model
2 USING(car_model_id) JOIN contract USING (car_id)
3
4
5 DELETE FROM contract
6 WHERE contract.car_id IN
7 (SELECT DISTINCT car.car_id FROM car JOIN car_model
8 ON car.car_model_id = car_model.car_model_id
9 WHERE car_model.release_year= (SELECT max(release_year)FROM car_model) )
10
11
```

Data Output Explain ×

The Data Output pane displays a table of car rental data. The columns are: car_price (money), milage (integer), model_name (character varying(30)), release_year (integer), rent_per_hour (numeric), technical_characteristics (character varying), and car_brand (character). The table has 7 rows, with the second row highlighted in blue. The data shows various car models like TOYOTA, JEEP CHEROKI, and MAZDA with their respective details.

	car_price money	milage integer	model_name character varying(30)	release_year integer	rent_per_hour numeric	technical_characteristics character varying	car_brand character
1	12,000.00 .₪.J	6000	TOYOTA	2015	800	[null]	Mazd
2	11,500.00 .₪.J	6000	JEEP CHEROKI	2019	1000	[null]	Jeep
3	12,000.00 .₪.J	6000	TOYOTA	2015	800	[null]	Mazd
4	11,500.00 .₪.J	6000	JEEP CHEROKI	2019	1000	[null]	Jeep
5	12,000.00 .₪.J	7000	JEEP CHEROKI	2019	1000	[null]	Jeep
6	12,000.00 .₪.J	6000	TOYOTA	2015	800	[null]	Mazd
7	11,500.00 .₪.J	6000	JEEP CHEROKI	2019	1000	[null]	Jeep

The screenshot shows the DataGrip IDE interface. The top navigation bar includes 'Properties', 'Dashboard', 'SQL', 'Statistics', 'Dependencies', 'Dependents', 'Processes', 'Car Rental/pos...', 'Untitled*', 'Car Rental/pos...', and a refresh icon. Below the navigation is a toolbar with icons for file operations, search, and help.

The main area has tabs for 'Messages', 'Query' (which is selected), 'Notifications', and 'Query History'. The 'Query' tab contains the following SQL code:

```
1 SELECT * FROM car JOIN car_model
2 USING(car_model_id) JOIN contract USING (car_id)
3
4
5 DELETE FROM contract
6 WHERE contract.car_id IN
7 (SELECT DISTINCT car.car_id FROM car JOIN car_model
8 ON car.car_model_id = car_model.car_model_id
9 WHERE car_model.release_year= (SELECT max(release_year)FROM car_model) )
10
11
```

Below the code is a 'Data Output' tab containing a table with three rows of data:

	car_price	milage	model_name	release_year	rent_per_hour	technical_characteristics
1	12,000.00	6000	TOYOTA	2015	800	[null]
2	12,000.00	6000	TOYOTA	2015	800	[null]
3	12,000.00	6000	TOYOTA	2015	800	[null]

At the bottom, status bars indicate 'Total rows: 3 of 3' and 'Query complete 00:00:00.937' on the left, and 'Ln 2, Col 49' on the right.

3- UPDATE

```
UPDATE contract SET car actual return date = CURRENT DATE WHERE contract id = 5
```

Properties Dashboard SQL Statistics Dependencies Dependents Processes Car Rental/pos... Untitled* Car Rental/pos...

Car Rental/postgres@PostgreSQL 15 No limit

Messages Query Notifications Query History

```

1 ALTER TABLE contract ALTER COLUMN rent_cost TYPE INTEGER USING rent_cost::integer
2 UPDATE contract SET rent_cost = CAST((CAST(rent_cost AS decimal)-100) AS MONEY)
3 WHERE car_id IN
4 (SELECT contract.car_id FROM insurance JOIN contract USING (insurance_id) WHERE insurance_type = 'Casual')
5
6
7 SELECT * FROM contract

```

Data Output Explain

	car_actual_return_date	is_extended	rent_hours	rent_cost	extension_hours	hours_late	is_return
1	[null]	false	70	3,900.00 . ⁰⁰ .J	[null]	[null]	false
2	[null]	false	40	4,900.00 . ⁰⁰ .J	[null]	[null]	false
3	[null]	false	40	4,900.00 . ⁰⁰ .J	[null]	30	false

Properties Dashboard SQL Statistics Dependencies Dependents Processes Car Rental/pos... Untitled* Car Rental/pos...

Car Rental/postgres@PostgreSQL 15 No limit

Messages Query Notifications Query History

```

1 rent_cost = CAST((CAST(rent_cost AS decimal)-100) AS MONEY)
2
3 id FROM insurance JOIN contract USING (insurance_id) WHERE insurance_type = 'Casual';
4
5 :t

```

Data Output Explain

	car_actual_return_date	is_extended	rent_hours	rent_cost	extension_hours	hours_late	is_return
1	[null]	false	70	3,800.00 . ⁰⁰ .J	[null]	[null]	false
2	[null]	false	40	4,800.00 . ⁰⁰ .J	[null]	[null]	false
3	[null]	false	40	4,800.00 . ⁰⁰ .J	[null]	30	false

3. Изучить графическое представление запросов и просмотреть историю запросов.

Properties Dashboard SQL Statistics Dependencies Dependents Processes Car Rental/pos... Untitled* Car Rental/pos... < > x

Car Rental/postgres@PostgreSQL 15

No limit

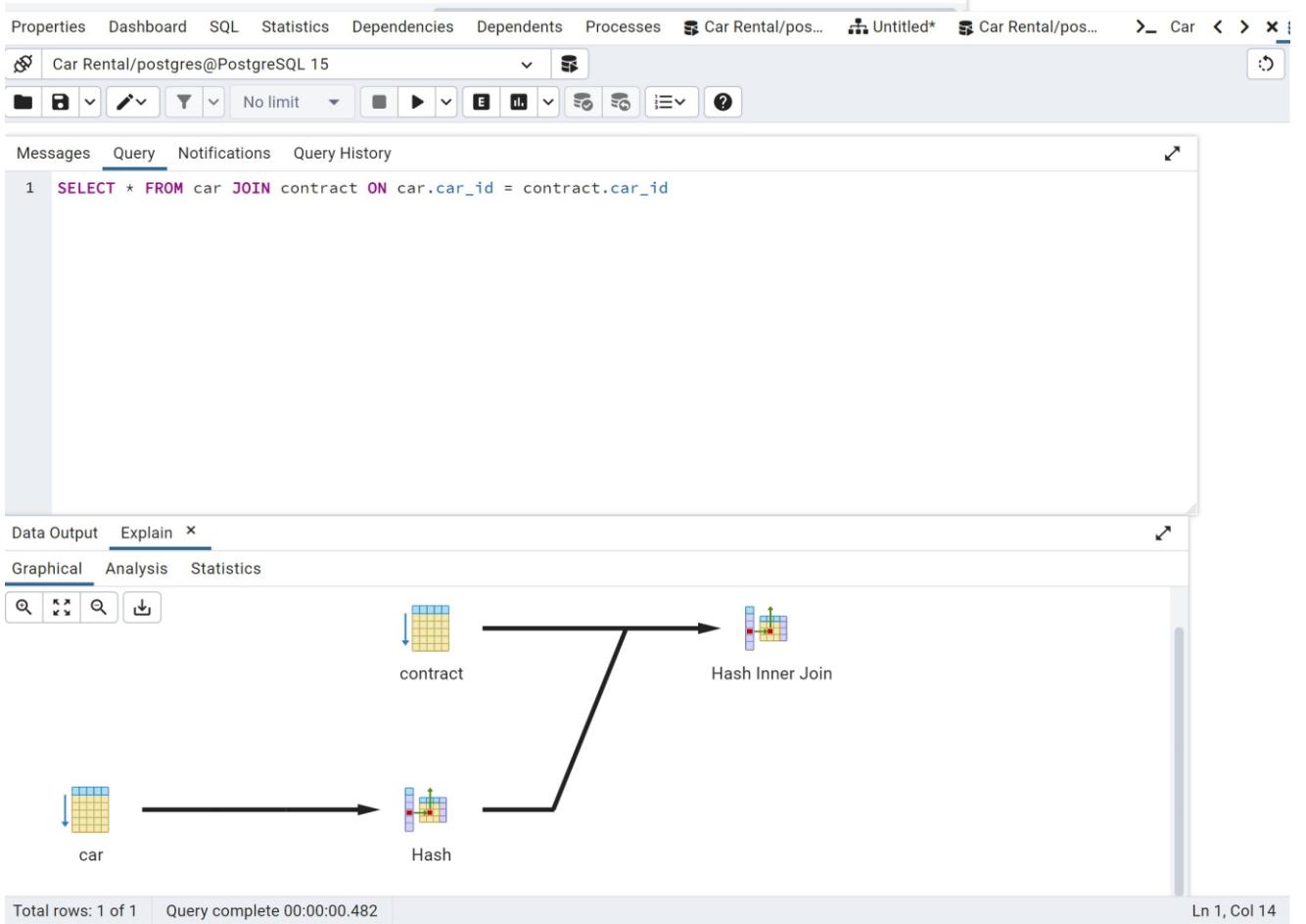
Messages Query Notifications Query History

```
1 UPDATE contract SET rent_cost = rent_cost - 100
2 WHERE car_id IN
3 (SELECT contract.car_id FROM insurance JOIN contract USING (insurance_id) WHERE insura
4
5
6
7 SELECT * FROM contract
```

Data Output Explain x

car_actual_return_date is_extended rent_hours rent_cost extension_hours hours_late is_return

	date	boolean	integer	money	integer	integer	boolean
1	[null]	false	70	4,000.00 .ω. .j	[null]	[null]	false
2	[null]	false	40	5,000.00 .ω. .j	[null]	[null]	false
3	[null]	false	40	5,000.00 .ω. .j	[null]	30	false



4. Создать простой и составной индексы для двух произвольных запросов и сравнить время выполнения запросов без индексов и с индексами. Для получения плана запроса использовать команду EXPLAIN.

The screenshot shows two separate sessions in pgAdmin 4. Both sessions have the same title bar: "Car Rental/postgres@PostgreSQL 15".

Session 1 (Top):

- Query Editor:** Contains the SQL query: `SELECT * FROM customer WHERE f_name = 'KAREEM';`
- Data Output:** Shows the result of the query, displaying one row from the "customer" table.
- Total rows: 1 of 1 Query complete 00:00:00.229**

customer_id [PK] integer	f_name character varying (30)	l_name character varying (30)	m_name character varying (30)	title character varying (30)	date_of_birth date	customer_type_id integer	passport_s character v
1	6 KAREEM	MOHSEN	BO AWF	AI expert	2001-02-15	1	033322534

Session 2 (Bottom):

- Query Editor:** Contains two SQL statements: `CREATE INDEX f_name_indx ON customer(f_name);` and `SELECT * FROM customer WHERE f_name = 'KAREEM';`
- Data Output:** Shows the result of the second query, displaying one row from the "customer" table.
- Total rows: 1 of 1 Query complete 00:00:00.109**

customer_id [PK] integer	f_name character varying (30)	l_name character varying (30)	m_name character varying (30)	title character varying (30)	date_of_birth date	customer_type_id integer	passport_s character v
1	6 KAREEM	MOHSEN	BO AWF	AI expert	2001-02-15	1	033322534

Индексирование сводит к минимуму время поиска, но оно может отнимать много времени в таблицах, в которые поступает много вставок.

Вывод:

В этой лабораторной работе я освоил команды DML SQL в postgresql и научился создавать сложные запросы и подзапросы.