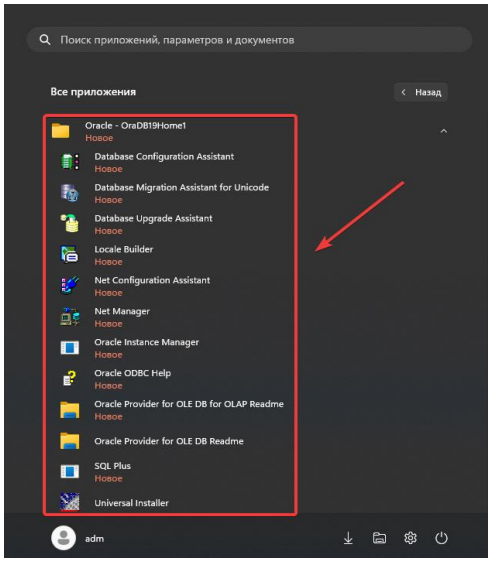
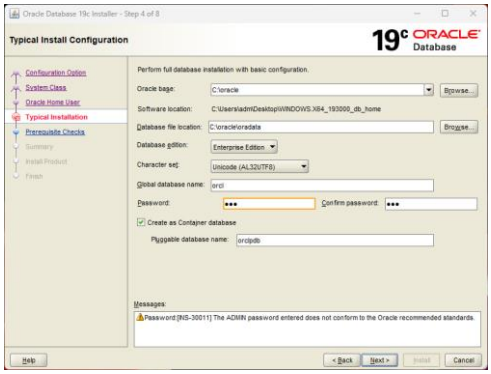
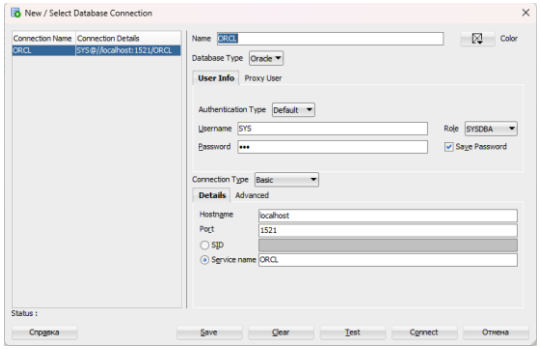
Тестовое задание.
Часть 1.

1. Установить Oracle Database Express Edition.



2. Разработать структуру и создать БД для сервиса сдачи автомобилей в аренду.





3. Основные сущности: клиенты, автомобили, заказы.

Учитывая, что используются только три сущности структура для сервиса сдачи автомобилей в аренду, будет выглядеть следующим образом:

Таблица Customers (Клиенты):
- customer_id (Уникальный идентификатор клиента);
- first_name (Имя клиента);
- last_name (Фамилия клиента);
- email (Адрес электронной почты клиента);
- phone (Номер телефона клиента);
- address (Адрес клиента).

Таблица Cars (Автомобили):
- car_id (Уникальный идентификатор автомобиля);
- make (Марка автомобиля);
- model (Модель автомобиля);
- year (Год выпуска автомобиля);
- registration (Регистрационный номер автомобиля);
- daily_rate (Дневная ставка аренды);
- status (Статус автомобиля, например, "доступен", "арендован").

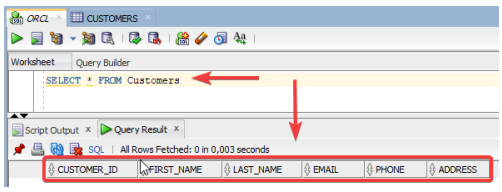
Таблица Orders (Заказы):
- order_id (Уникальный идентификатор заказа);
- customer_id (Ссылка на клиента, сделавшего заказ);
- car_id (Ссылка на арендованный автомобиль);
- start_date (Дата начала аренды);
- end_date (Дата окончания аренды);
- total_cost (Итоговая стоимость аренды).

Следующим шагом создадим таблицу Customers, в соответствии с условием:

```
CREATE TABLE Customers (
    customer_id NUMBER PRIMARY KEY,
    first_name VARCHAR2(50),
    last_name VARCHAR2(50),
    email VARCHAR2(100),
    phone VARCHAR2(20),
    address VARCHAR2(255)
);
```

Проверим создание таблицы:

```
SELECT * FROM Customers
```



Создадим аналогичным образом таблицу Cars, в соответствии с условием:

```
CREATE TABLE Cars (
    car_id NUMBER PRIMARY KEY,
    make VARCHAR2(50),
    model VARCHAR2(50),
    year NUMBER,
    registration VARCHAR2(20) UNIQUE,
    daily_rate NUMBER,
    status VARCHAR2(20)
);
```

Добавлено примечание ([a1]): CREATE TABLE Customers: Эта часть SQL-команды начинает процесс создания новой таблицы с именем "Customers". Она определяет начало определения таблицы.

(customer_id NUMBER PRIMARY KEY,: Здесь мы определяем первое поле "customer_id". NUMBER - это тип данных, который предназначен для хранения числовых значений, в данном случае, идентификаторов клиентов. PRIMARY KEY указывает, что это поле является первичным ключом, что означает, что значения в этом столбце должны быть уникальными.

first_name VARCHAR2(50),: Это определение второго поля "first_name". VARCHAR2(50) означает, что это текстовое поле переменной длины (VARCHAR) с максимальной длиной в 50 символов. Это поле будет содержать имена клиентов.

last_name VARCHAR2(50),: Это определение третьего поля "last_name". То же, что и для "first_name", это текстовое поле с максимальной длиной в 50 символов, предназначенное для фамилий клиентов.

email VARCHAR2(100),: Это четвертое поле "email". Также текстовое поле переменной длины с максимальной длиной в 100 символов, предназначенное для хранения адресов электронной почты клиентов.

phone VARCHAR2(20),: Это определение пятого поля "phone". Это текстовое поле переменной длины с ...

Добавлено примечание ([a2]): Команда SELECT * FROM Customers - это SQL-команда, используемая для извлечения данных из таблицы "Customers" в базе данных.

SELECT: Это ключевое слово, которое указывает базе данных на необходимость выполнения операции выборки данных.

*: Звездочка (*) означает "все столбцы". Это обозначение говорит СУБД о том, что мы хотим выбрать все столбцы, содержащиеся в таблице "Customers".

FROM: Это ключевое слово, за которым следует имя таблицы, из которой мы хотим извлечь данные. В данном случае, имя таблицы - "Customers".

Добавлено примечание ([a3]): CREATE TABLE Cars: Эта часть SQL-команды начинает процесс создания новой таблицы с именем "Cars". Она определяет начало определения таблицы.

(car_id NUMBER PRIMARY KEY,: Здесь мы определяем первое поле "car_id". NUMBER - это тип данных, предназначенный для хранения числовых значений, в данном случае, идентификаторов автомобилей. PRIMARY KEY указывает, что это поле является первичным ключом, что гарантирует уникальность идентификаторов автомобилей в таблице.

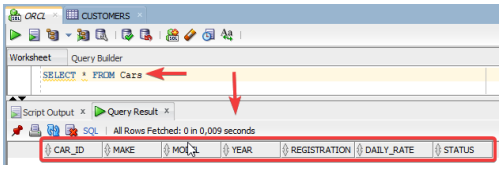
make VARCHAR2(50),: Это определение второго поля "make". VARCHAR2(50) означает, что это текстовое поле переменной длины (VARCHAR) с максимальной длиной в 50 символов. Это поле будет содержать марку (бренд) автомобиля.

model VARCHAR2(50),: Это определение третьего поля "model". То же, что и для "make", это текстовое поле с максимальной длиной в 50 символов, предназначенное для модели автомобиля. ...

Проверим создание таблицы:

```
SELECT * FROM Cars
```

Добавлено примечание ([a4]): Тоже что и команда SELECT * FROM Customers



Создадим таблицу Orders с внешними ключами, что позволит хранить данные о заказах и связывать их с соответствующими клиентами и автомобилями:

```
CREATE TABLE Orders (
    order_id NUMBER PRIMARY KEY,
    customer_id NUMBER,
    car_id NUMBER,
    start_date DATE,
    end_date DATE,
    total_cost NUMBER,
    CONSTRAINT fk_customer
        FOREIGN KEY (customer_id)
        REFERENCES Customers(customer_id),
    CONSTRAINT fk_car
        FOREIGN KEY (car_id)
        REFERENCES Cars(car_id)
);
```

Добавлено примечание ([a5]): CREATE TABLE Orders: Эта часть команды начинает процесс создания новой таблицы с именем "Orders".

order_id NUMBER PRIMARY KEY: Здесь определены столбцы таблицы.

order_id - это столбец с числовыми значениями, который будет использоваться в качестве первичного ключа таблицы. Первичный ключ уникален для каждой записи и обеспечивает уникальность каждого заказа в таблице. NUMBER - это тип данных столбца. В данном случае, это числовой тип данных.
customer_id NUMBER и car_id NUMBER: Эти столбцы также представляют числовые значения и будут использоваться для хранения идентификаторов клиента и автомобиля, связанных с каждым заказом.

start_date DATE и end_date DATE: Эти столбцы хранят дату начала и окончания аренды автомобиля. Тип данных DATE используется для хранения дат.

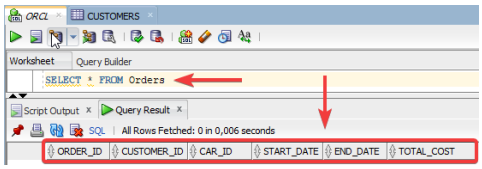
total_cost NUMBER: Этот столбец предназначен для хранения итоговой стоимости заказа и также имеет числовой тип данных.

CONSTRAINT fk_customer FOREIGN KEY (customer_id) REFERENCES Customers(customer_id): Эта часть команды определяет внешний ключ для столбца customer_id, который связывает столбец customer_id в таблице "Orders" с первичным ключом customer_id в таблице "Customers". Это обеспечивает целостность данных и позволяет связывать заказы с клиентами.

CONSTRAINT fk_car FOREIGN KEY (car_id) REFERENCES Cars(car_id): Аналогично, это определяет внешний ключ для столбца car_id, который связывает столбец car_id в таблице "Orders" с первичным ключом car_id в таблице "Cars". Это позволяет связывать заказы с автомобилями.

Проверим создание таблицы:

```
SELECT * FROM Orders
```



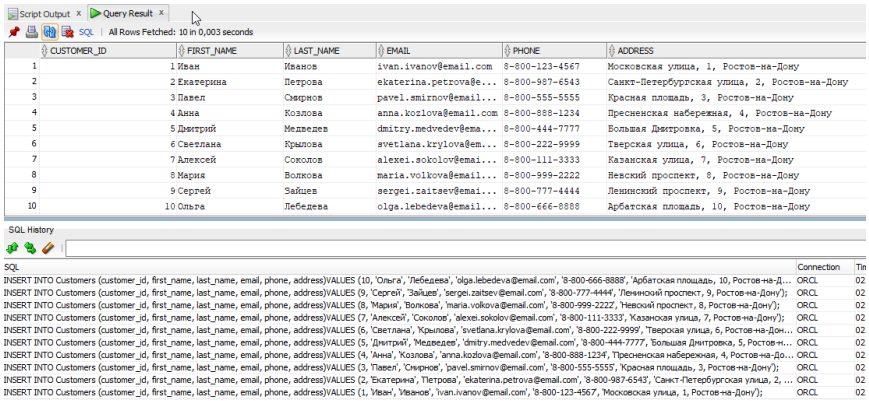
Добавлено примечание ([a6]): Тоже что и команда SELECT * FROM Customers

4. Реализовать:

    a. Скрипты для наполнения БД тестовыми данными.

Для наполнения базы данных тестовыми данными в таблицу Customers внесем следующие данные:

```
INSERT INTO Customers (customer_id, first_name, last_name, email, phone, address)
VALUES (1, 'Иван', 'Иванов', 'ivan.ivanov@email.com', '8-800-123-4567', 'Московская улица, 1, Ростов-на-Дону');

INSERT INTO Customers (customer_id, first_name, last_name, email, phone, address)
VALUES (2, 'Екатерина', 'Петрова', 'ekaterina.petrova@email.com', '8-800-987-6543', 'Санкт-Петербургская улица, 2, Ростов-на-Дону');

INSERT INTO Customers (customer_id, first_name, last_name, email, phone, address)
VALUES (3, 'Павел', 'Смирнов', 'pavel.smirnov@email.com', '8-800-555-5555', 'Красная площадь, 3, Ростов-на-Дону');

INSERT INTO Customers (customer_id, first_name, last_name, email, phone, address)
VALUES (4, 'Анна', 'Козлова', 'anna.kozlova@email.com', '8-800-888-1234', 'Пресненская набережная, 4, Ростов-на-Дону');

INSERT INTO Customers (customer_id, first_name, last_name, email, phone, address)
VALUES (5, 'Дмитрий', 'Медведев', 'dmitry.medvedev@email.com', '8-800-444-7777', 'Большая Дмитровка, 5, Ростов-на-Дону');

INSERT INTO Customers (customer_id, first_name, last_name, email, phone, address)
VALUES (6, 'Светлана', 'Крылова', 'svetlana.krylova@email.com', '8-800-222-9999', 'Тверская улица, 6, Ростов-на-Дону');

INSERT INTO Customers (customer_id, first_name, last_name, email, phone, address)
VALUES (7, 'Алексей', 'Соколов', 'alexei.sokolov@email.com', '8-800-111-3333', 'Казанская улица, 7, Ростов-на-Дону');

INSERT INTO Customers (customer_id, first_name, last_name, email, phone, address)
VALUES (8, 'Мария', 'Волкова', 'maria.volkova@email.com', '8-800-999-2222', 'Невский проспект, 8, Ростов-на-Дону');

INSERT INTO Customers (customer_id, first_name, last_name, email, phone, address)
VALUES (9, 'Сергей', 'Зайцев', 'sergei.zaitsev@email.com', '8-800-777-4444', 'Ленинский проспект, 9, Ростов-на-Дону');

INSERT INTO Customers (customer_id, first_name, last_name, email, phone, address)
VALUES (10, 'Ольга', 'Лебедева', 'olga.lebedeva@email.com', '8-800-666-8888', 'Арбатская площадь, 10, Ростов-на-Дону');
```



Script Output | Query Result

SQL | All Rows Fetched: 10 in 0,003 seconds

	CUSTOMER_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE	ADDRESS
1	1	Иван	Иванов	ivan.ivanov@email.com	8-800-123-4567	Московская улица, 1, Ростов-на-Дону
2	2	Екатерина	Петрова	ekaterina.petrova@e...	8-800-987-6543	Санкт-Петербургская улица, 2, Ростов-на-Дону
3	3	Павел	Смирнов	pavel.smirnov@email...	8-800-555-5555	Красная площадь, 3, Ростов-на-Дону
4	4	Анна	Козлова	anna.kozlova@email.com	8-800-888-1234	Пресненская набережная, 4, Ростов-на-Дону
5	5	Дмитрий	Медведев	dmitry.medvedev@ema...	8-800-444-7777	Большая Дмитровка, 5, Ростов-на-Дону
6	6	Светлана	Крылова	svetlana.krylova@em...	8-800-222-9999	Тверская улица, 6, Ростов-на-Дону
7	7	Алексей	Соколов	alexei.sokolov@emai...	8-800-111-3333	Казанская улица, 7, Ростов-на-Дону
8	8	Мария	Волкова	maria.volkova@email...	8-800-999-2222	Невский проспект, 8, Ростов-на-Дону
9	9	Сергей	Зайцев	sergei.zaitsev@emai...	8-800-777-4444	Ленинский проспект, 9, Ростов-на-Дону
10	10	Ольга	Лебедева	olga.lebedeva@email...	8-800-666-8888	Арбатская площадь, 10, Ростов-на-Дону

SQL History

SQL	Connection	Tim
INSERT INTO Customers (customer_id, first_name, last_name, email, phone, address)VALUES (10, 'Ольга', 'Лебедева', 'olga.lebedeva@email.com', '8-800-666-8888', 'Арбатская площадь, 10, Ростов-на-Д...	ORCL	02
INSERT INTO Customers (customer_id, first_name, last_name, email, phone, address)VALUES (9, 'Сергей', 'Зайцев', 'sergei.zaitsev@email.com', '8-800-777-4444', 'Ленинский проспект, 9, Ростов-на-Дону');	ORCL	02
INSERT INTO Customers (customer_id, first_name, last_name, email, phone, address)VALUES (8, 'Мария', 'Волкова', 'maria.volkova@email.com', '8-800-999-2222', 'Невский проспект, 8, Ростов-на-Дону');	ORCL	02
INSERT INTO Customers (customer_id, first_name, last_name, email, phone, address)VALUES (7, 'Алексей', 'Соколов', 'alexei.sokolov@email.com', '8-800-111-3333', 'Казанская улица, 7, Ростов-на-Дону');	ORCL	02
INSERT INTO Customers (customer_id, first_name, last_name, email, phone, address)VALUES (6, 'Светлана', 'Крылова', 'svetlana.krylova@email.com', '8-800-222-9999', 'Тверская улица, 6, Ростов-на-Дон...	ORCL	02
INSERT INTO Customers (customer_id, first_name, last_name, email, phone, address)VALUES (5, 'Дмитрий', 'Медведев', 'dmitry.medvedev@email.com', '8-800-444-7777', 'Большая Дмитровка, 5, Ростов-н...	ORCL	02
INSERT INTO Customers (customer_id, first_name, last_name, email, phone, address)VALUES (4, 'Анна', 'Козлова', 'anna.kozlova@email.com', '8-800-888-1234', 'Пресненская набережная, 4, Ростов-на-До...	ORCL	02
INSERT INTO Customers (customer_id, first_name, last_name, email, phone, address)VALUES (3, 'Павел', 'Смирнов', 'pavel.smirnov@email.com', '8-800-555-5555', 'Красная площадь, 3, Ростов-на-Дону');	ORCL	02
INSERT INTO Customers (customer_id, first_name, last_name, email, phone, address)VALUES (2, 'Екатерина', 'Петрова', 'ekaterina.petrova@email.com', '8-800-987-6543', 'Санкт-Петербургская улица, 2, ...	ORCL	02
INSERT INTO Customers (customer_id, first_name, last_name, email, phone, address)VALUES (1, 'Иван', 'Иванов', 'ivan.ivanov@email.com', '8-800-123-4567', 'Московская улица, 1, Ростов-на-Дону');	ORCL	02

Добавлено примечание ([a7]): Для наполнения базы данных тестовыми данными с использованием SQL, мы будем использовать команды INSERT INTO.

Синтаксис команды INSERT INTO:

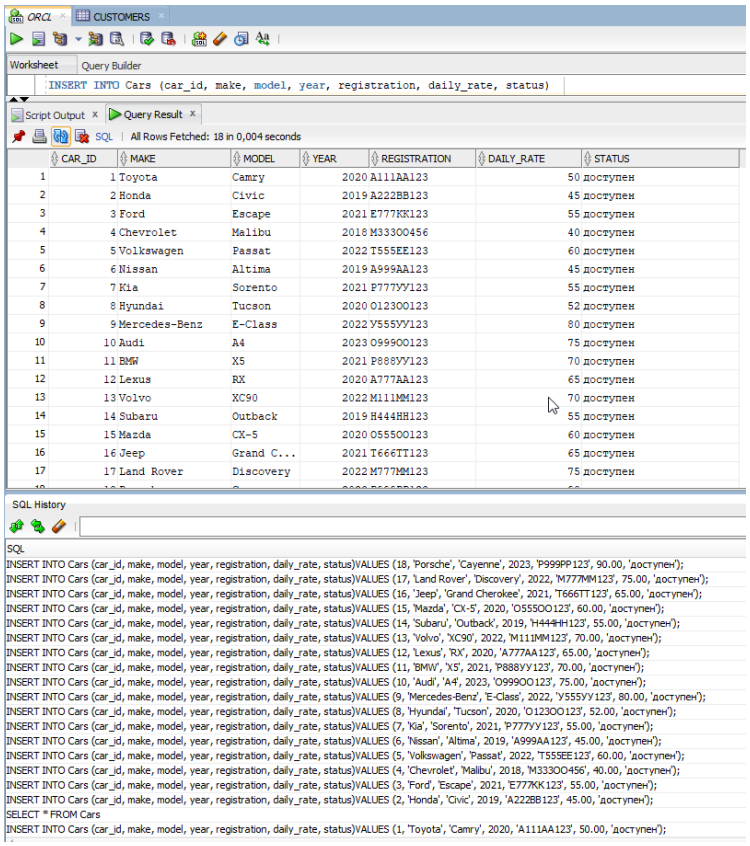
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);

Где:
INSERT INTO: Это ключевое слово, которое указывает на начало операции вставки данных в таблицу.
table_name: Это имя таблицы, в которую мы хотим вставить данные.
(column1, column2, column3, ...): Это перечисление столбцов, в которые мы хотим вставить данные. Мы можем указать только те столбцы, которые нужно заполнить, остальные столбцы могут иметь значения по умолчанию или быть NULL, если не указаны.
VALUES (value1, value2, value3, ...): Это перечисление значений, которые мы хотим вставить в соответствующие столбцы. Порядок значений должен соответствовать порядку столбцов.

Для наполнения базы данных тестовыми данными в таблицу Cars внесем следующие данные:

```
INSERT INTO Cars (car_id, make, model, year, registration, daily_rate, status)
VALUES (1, 'Toyota', 'Camry', 2020, 'A111AA123', 50.00, 'доступен');

INSERT INTO Cars (car_id, make, model, year, registration, daily_rate, status)
VALUES (2, 'Honda', 'Civic', 2019, 'A222BB123', 45.00, 'доступен');

INSERT INTO Cars (car_id, make, model, year, registration, daily_rate, status)
VALUES (3, 'Ford', 'Escape', 2021, 'E777KK123', 55.00, 'доступен');

INSERT INTO Cars (car_id, make, model, year, registration, daily_rate, status)
VALUES (4, 'Chevrolet', 'Malibu', 2018, 'M333OO456', 40.00, 'доступен');

INSERT INTO Cars (car_id, make, model, year, registration, daily_rate, status)
VALUES (5, 'Volkswagen', 'Passat', 2022, 'T555EE123', 60.00, 'доступен');

INSERT INTO Cars (car_id, make, model, year, registration, daily_rate, status)
VALUES (6, 'Nissan', 'Altima', 2019, 'A999AA123', 45.00, 'доступен');

INSERT INTO Cars (car_id, make, model, year, registration, daily_rate, status)
VALUES (7, 'Kia', 'Sorento', 2021, 'P777УУ123', 55.00, 'доступен');

INSERT INTO Cars (car_id, make, model, year, registration, daily_rate, status)
VALUES (8, 'Hyundai', 'Tucson', 2020, 'O123OO123', 52.00, 'доступен');

INSERT INTO Cars (car_id, make, model, year, registration, daily_rate, status)
VALUES (9, 'Mercedes-Benz', 'E-Class', 2022, 'У555УУ123', 80.00, 'доступен');

INSERT INTO Cars (car_id, make, model, year, registration, daily_rate, status)
VALUES (10, 'Audi', 'A4', 2023, 'O999OO123', 75.00, 'доступен');

INSERT INTO Cars (car_id, make, model, year, registration, daily_rate, status)
VALUES (11, 'BMW', 'X5', 2021, 'P888УУ123', 70.00, 'доступен');

INSERT INTO Cars (car_id, make, model, year, registration, daily_rate, status)
VALUES (12, 'Lexus', 'RX', 2020, 'A777AA123', 65.00, 'доступен');

INSERT INTO Cars (car_id, make, model, year, registration, daily_rate, status)
VALUES (13, 'Volvo', 'XC90', 2022, 'M111MM123', 70.00, 'доступен');

INSERT INTO Cars (car_id, make, model, year, registration, daily_rate, status)
VALUES (14, 'Subaru', 'Outback', 2019, 'H444HH123', 55.00, 'доступен');

INSERT INTO Cars (car_id, make, model, year, registration, daily_rate, status)
VALUES (15, 'Mazda', 'CX-5', 2020, 'O555OO123', 60.00, 'доступен');

INSERT INTO Cars (car_id, make, model, year, registration, daily_rate, status)
VALUES (16, 'Jeep', 'Grand Cherokee', 2021, 'T666TT123', 65.00, 'доступен');

INSERT INTO Cars (car_id, make, model, year, registration, daily_rate, status)
VALUES (17, 'Land Rover', 'Discovery', 2022, 'M777MM123', 75.00, 'доступен');

INSERT INTO Cars (car_id, make, model, year, registration, daily_rate, status)
VALUES (18, 'Porsche', 'Cayenne', 2023, 'P999PP123', 90.00, 'доступен');
```

Добавлено примечание ([a8]): Тоже что и команда INSERT INTO для Customers

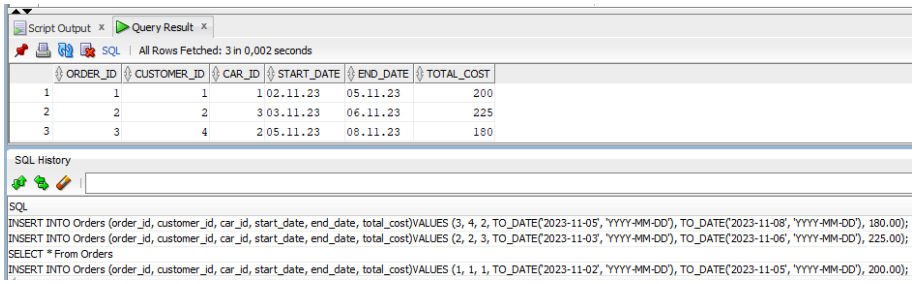


Для наполнения базы данных тестовыми данными в таблицу Orders внесем следующие данные:

INSERT INTO Orders (order_id, customer_id, car_id, start_date, end_date, total_cost)
VALUES (1, 1, 1, TO_DATE('2023-11-02', 'YYYY-MM-DD'), TO_DATE('2023-11-05', 'YYYY-MM-DD'), 200.00);

INSERT INTO Orders (order_id, customer_id, car_id, start_date, end_date, total_cost)
VALUES (2, 2, 3, TO_DATE('2023-11-03', 'YYYY-MM-DD'), TO_DATE('2023-11-06', 'YYYY-MM-DD'), 225.00);

INSERT INTO Orders (order_id, customer_id, car_id, start_date, end_date, total_cost)
VALUES (3, 4, 2, TO_DATE('2023-11-05', 'YYYY-MM-DD'), TO_DATE('2023-11-08', 'YYYY-MM-DD'), 180.00);



Добавлено примечание ([a9]): INSERT INTO: Ключевое слово INSERT INTO указывает на то, что вы собираетесь вставить новую строку данных в таблицу.

Table Name: Это имя таблицы, в которую мы хотим вставить новую строку данных.

Column Names: Если мы явно указываем столбцы, в которые мы собираемся вставить данные, то после имени таблицы следует перечислить имена столбцов в скобках.

VALUES: Ключевое слово VALUES указывает на начало списка значений, которые мы собираемся вставить в соответствующие столбцы.

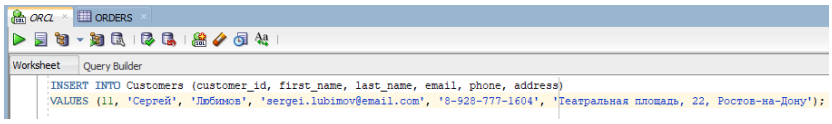
Data Values: Значения, которые мы вставляем, должны соответствовать типам данных столбцов, в которые они вставляются. Если столбцы имеют числовые значения, то числовые значения должны быть представлены без кавычек, а если столбцы имеют строковые значения, то строковые значения должны быть заключены в одинарные кавычки (' ').

Дата и время: Если столбцы таблицы имеют тип данных даты и времени, то даты должны быть представлены в формате, который соответствует формату даты и времени в нашей базе данных. Используется функция TO_DATE для преобразования строкового значения в тип данных даты.

b. CRUD операции для всех сущностей.

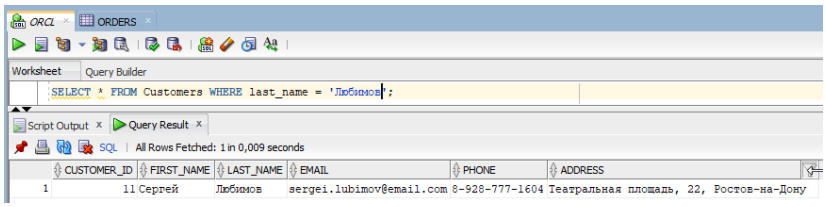
CRUD операции **для сущности Customers**:
Create (Создание):

INSERT INTO Customers (customer_id, first_name, last_name, email, phone, address)
VALUES (11, 'Сергей', 'Любимов', 'sergei.lubimov@email.com', '8-928-777-1604', 'Театральная площадь, 22, Ростов-на-Дону');



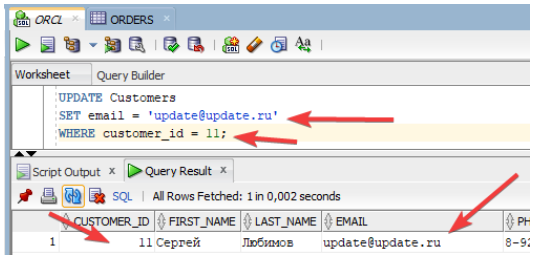
Read (Чтение):

*SELECT * FROM Customers WHERE last_name = 'Любимов';*



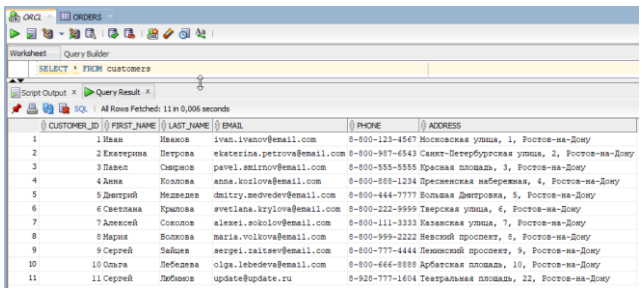
Update (Обновление):

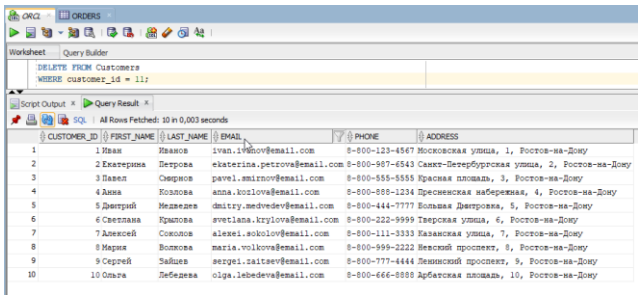
UPDATE Customers
SET email = 'update@update.ru'
WHERE customer_id = 11;



Delete (Удаление):

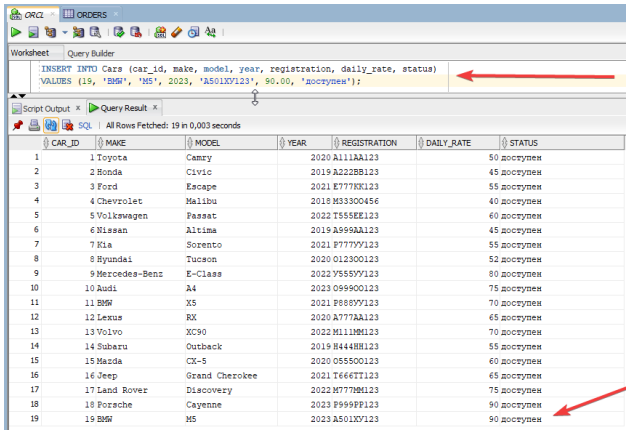
DELETE FROM Customers
WHERE customer_id = 11;





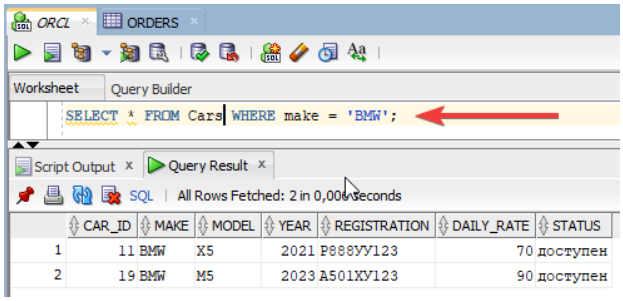
CRUD операции **для сущности Cars:**
Create (Создание):

INSERT INTO Cars (car_id, make, model, year, registration, daily_rate, status)
VALUES (19, 'BMW', 'M5', 2023, 'A501XY123', 90.00, 'доступен');



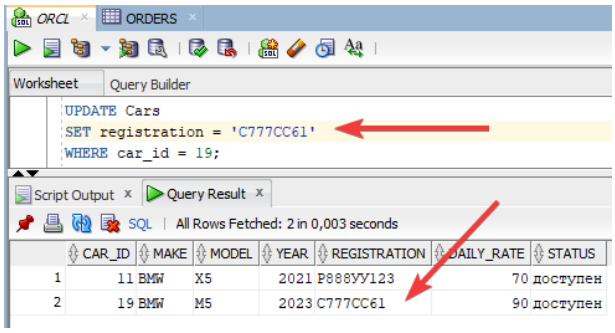
Read (Чтение):

*SELECT * FROM Customers WHERE make = 'BMW';*



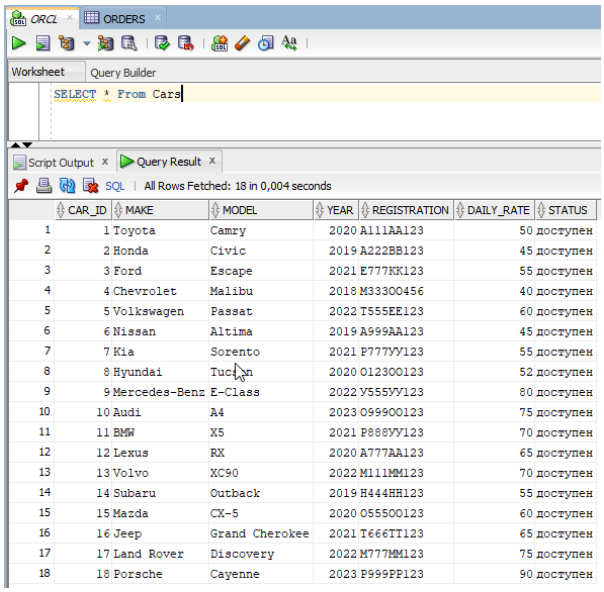
Update (Обновление):

UPDATE Cars
SET registration = C777CC61
WHERE car_id = 19;



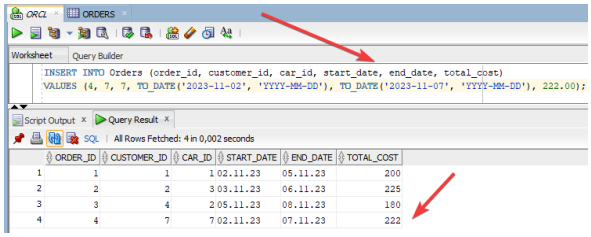
Delete (Удаление):

DELETE FROM Cars
WHERE car_id = 19;



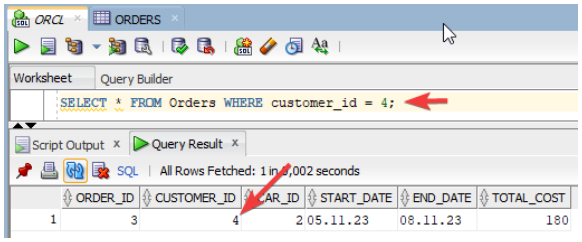
CRUD операции **для сущности Orders:**
Create (Создание):

INSERT INTO Orders (order_id, customer_id, car_id, start_date, end_date, total_cost)
VALUES (4, 7, 7, TO_DATE('2023-11-02', 'YYYY-MM-DD'), TO_DATE('2023-11-07', 'YYYY-MM-DD'), 222.00);



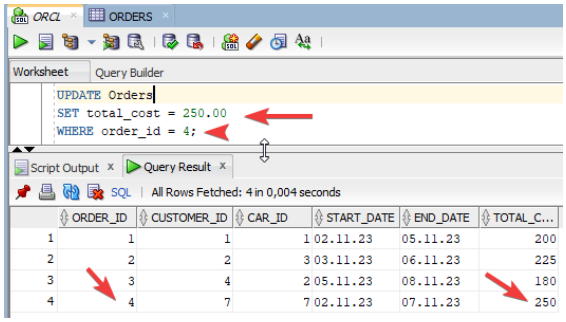
Read (Чтение):

*SELECT * FROM Orders WHERE customer_id = 4;*



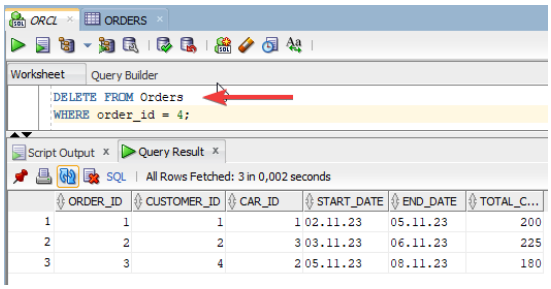
Update (Обновление):

UPDATE Orders
SET total_cost = 250.00
WHERE order_id = 4;



Delete (Удаление):

DELETE FROM Orders
WHERE order_id = 4;



c. сохранение истории изменения заказов.

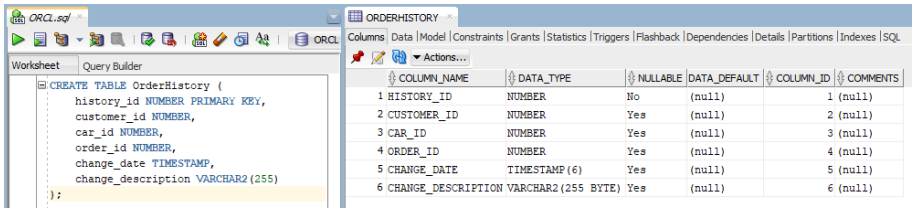
Для сохранения истории изменений заказов в базе данных нам потребуется создать таблицу для хранения истории изменений, создать последовательности и триггер, который будет записывать изменения в эту таблицу:

Создание таблицы:

```
CREATE TABLE OrderHistory (
    history_id NUMBER PRIMARY KEY,
    customer_id NUMBER,
    car_id NUMBER,
    order_id NUMBER,
    change_date TIMESTAMP,
    change_description VARCHAR2(255)
);
```

Добавлено примечание ([a10]): Таблица "OrderHistory," будет хранить историю изменений заказов.

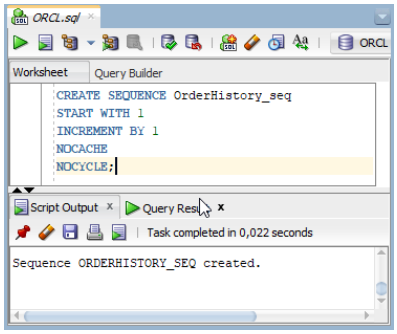
history_id (уникальный идентификатор записи в истории, первичный ключ).
customer_id (ссылка на клиента, совершившего заказ).
car_id (ссылка на автомобиль в заказе).
order_id (ссылка на исходный заказ).
change_date (дата и время изменения).
change_description (описание изменения заказа, например, "создание заказа," "обновление заказа").



Создание последовательности:

```
CREATE SEQUENCE OrderHistory_seq
START WITH 1
INCREMENT BY 1
NOCACHE
NOCYCLE;
```

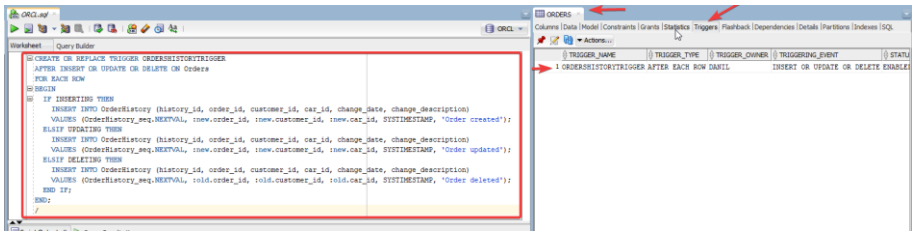
Добавлено примечание ([a11]): Последовательность (sequence), которая будет использоваться для генерации уникальных значений для столбца history_id в таблице "OrderHistory."



Создание триггера для отслеживания изменений:

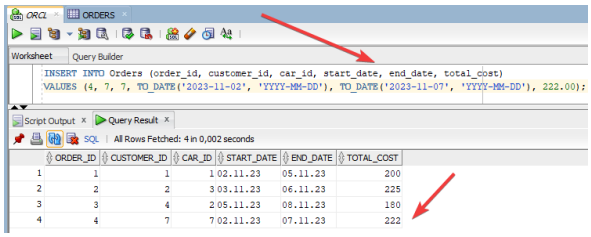
```
CREATE OR REPLACE TRIGGER ORDERSHISTORYTRIGGER
AFTER INSERT OR UPDATE OR DELETE ON Orders
FOR EACH ROW
BEGIN
  IF INSERTING THEN
    INSERT INTO OrderHistory (history_id, order_id, customer_id, car_id, change_date, change_description)
    VALUES (OrderHistory_seq.NEXTVAL, :new.order_id, :new.customer_id, :new.car_id, SYSTIMESTAMP,
'Order created');
  ELSIF UPDATING THEN
    INSERT INTO OrderHistory (history_id, order_id, customer_id, car_id, change_date, change_description)
    VALUES (OrderHistory_seq.NEXTVAL, :new.order_id, :new.customer_id, :new.car_id, SYSTIMESTAMP,
'Order updated');
```

```
 ELSIF DELETING THEN
  INSERT INTO OrderHistory (history_id, order_id, customer_id, car_id, change_date, change_description)
  VALUES (OrderHistory_seq.NEXTVAL, :old.order_id, :old.customer_id, :old.car_id, SYSTIMESTAMP,
'Order deleted');
 END IF;
END;
/
```

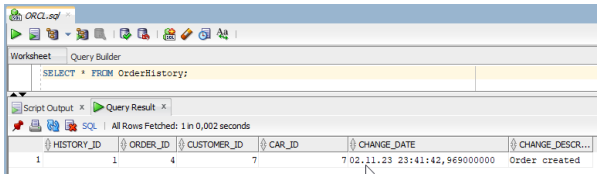


Чтобы проверить сохранение истории заказов добавим информацию об аренде:

```
INSERT INTO Orders (order_id, customer_id, car_id, start_date, end_date, total_cost)
VALUES (4, 7, 7, TO_DATE('2023-11-02', 'YYYY-MM-DD'), TO_DATE('2023-11-07', 'YYYY-MM-DD'),
222.00);
```



Убедимся, что изменения, сделанные в таблице Orders отображены в таблице OrderHistory:



Добавлено примечание ([a12]): CREATE OR REPLACE TRIGGER ORDERSHISTORYTRIGGER - это начало команды создания триггера. Она начинается с ключевого слова CREATE, что указывает на создание нового триггера. Фраза OR REPLACE указывает, что если триггер с таким именем уже существует, он будет перезаписан новым триггером с тем же именем. ORDERSHISTORYTRIGGER - это имя нашего триггера.

AFTER INSERT OR UPDATE OR DELETE ON Orders - это часть, которая определяет, когда триггер будет срабатывать. В данном случае, триггер срабатывает после операций вставки (INSERT), обновления (UPDATE) и удаления (DELETE) в таблице Orders. Таким образом, триггер будет реагировать на любые изменения в нашей таблице.

FOR EACH ROW - это часть, которая указывает, что триггер будет выполняться для каждой строки, затронутой операцией. Внутри блока триггера можно обращаться к данным строки, которую затронула операция.

BEGIN - начало блока триггера. Все операции триггера размещаются между BEGIN и END.

Блок IF INSERTING THEN ... ELSIF UPDATING THEN ... ELSIF DELETING THEN ... END IF; - это условные операторы, которые определяют, какое действие было выполнено (вставка, обновление или удаление). В зависимости от действия выполняется соответствующая операция записи в таблицу OrderHistory.

В каждом блоке IF используются операторы INSERT INTO, чтобы добавить запись в таблицу OrderHistory. Каждая запись содержит следующие значения:

history_id - значение, которое генерируется из последовательности OrderHistory_seq.
customer_id, car_id, order_id - значения, полученные из данных строки, которую затронула операция.
change_date - текущая дата и время, полученные с помощью функции SYSTIMESTAMP.
change_description - описание изменения (например, "Order created," "Order updated," "Order deleted").
/ - это окончание команды.

d. отчет об истории изменения заказов.

Выполним запрос:

```
SELECT
    oh.history_id,
    oh.change_date,
    o.order_id,
    o.customer_id,
    o.car_id,
    oh.change_description
FROM
    OrderHistory oh
JOIN
    Orders o ON oh.order_id = o.order_id
ORDER BY
    oh.change_date DESC;
```

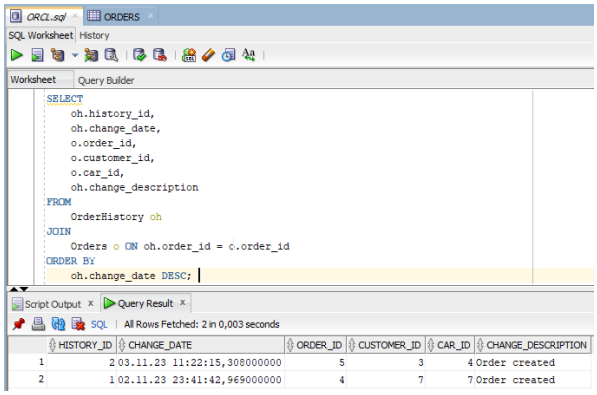
Добавлено примечание ([a13]): SQL-запрос выбирает столбцы из таблицы "OrderHistory" и "Orders," включая history_id, change_date, order_id, customer_id, car_id и change_description.

Использует оператор JOIN, чтобы связать таблицы "OrderHistory" и "Orders" по полю order_id, чтобы получить информацию о заказе, связанном с каждой записью истории.

Определяет порядок сортировки результатов по столбцу change_date в убывающем порядке (от самых новых изменений к более старым).

Чтобы проверить работу отчета об истории изменения заказов добавим информацию об аренде:

```
INSERT INTO Orders (order_id, customer_id, car_id, start_date, end_date, total_cost)
VALUES (5, 3, 4, TO_DATE('2023-11-03', 'YYYY-MM-DD'), TO_DATE('2023-11-08', 'YYYY-MM-DD'),
100.00);
```

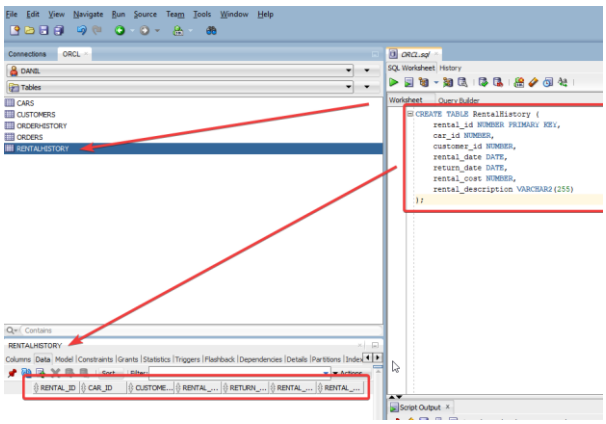


e. сохранение истории сдачи в аренду авто.

Для реализации сохранения истории сдачи автомобилей в аренду мы создадим таблицу «RentalHistory» и создадим триггер для записи изменений в эту таблицу:

Создание таблицы для истории аренды:

```
CREATE TABLE RentalHistory (
    rental_id NUMBER PRIMARY KEY,
    car_id NUMBER,
    customer_id NUMBER,
    rental_date DATE,
    return_date DATE,
    rental_cost NUMBER,
    rental_description VARCHAR2(255)
);
```



Добавлено примечание ([a14]): rental_id: Это уникальный идентификатор для каждой записи в таблице "RentalHistory". Используется для идентификации конкретной аренды.

car_id: Этот столбец содержит идентификатор автомобиля, который сдается в аренду.

customer_id: Здесь хранится идентификатор клиента, сдающего автомобиль в аренду.

rental_date: Этот столбец представляет дату начала аренды автомобиля.

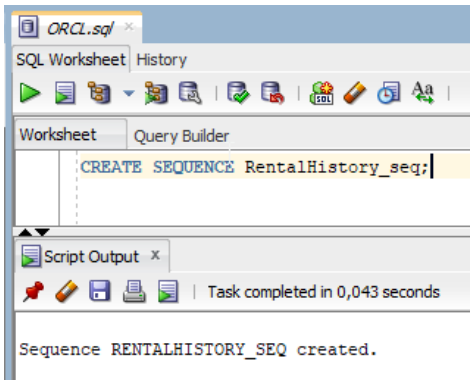
return_date: Здесь записывается дата окончания аренды.

rental_cost: В этом столбце можно сохранить информацию о стоимости аренды автомобиля.

rental_description: Этот столбец позволяет добавить описание или комментарий к событию сдачи в аренду.

Создание последовательности «RentalHistory_seq», ее мы будем использовать для генерации уникальных идентификаторов:

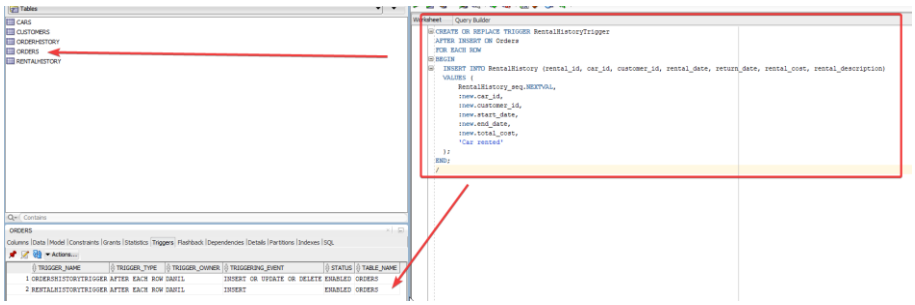
```
CREATE SEQUENCE RentalHistory_seq;
```



Создадим триггер для записи сдачи аренды авто:

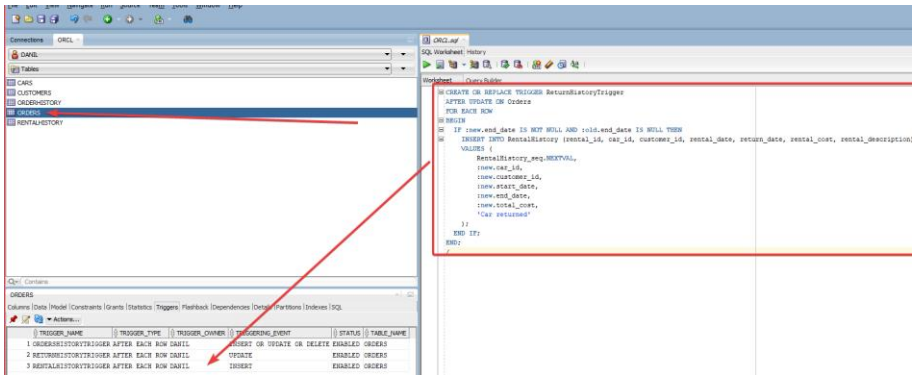
```
CREATE OR REPLACE TRIGGER RentalHistoryTrigger
AFTER INSERT ON Orders
FOR EACH ROW
BEGIN
```

```
 INSERT INTO RentalHistory (rental_id, car_id, customer_id, rental_date, return_date, rental_cost,
rental_description)
 VALUES (
    RentalHistory_seq.NEXTVAL,
    :new.car_id,
    :new.customer_id,
    :new.start_date,
    :new.end_date,
    :new.total_cost,
    'Car rented'
 );
END;
/
```



Создадим триггер для записи возврата аренды авто:

```
CREATE OR REPLACE TRIGGER ReturnHistoryTrigger
AFTER UPDATE ON Orders
FOR EACH ROW
BEGIN
 IF :new.end_date IS NOT NULL AND :old.end_date IS NULL THEN
   INSERT INTO RentalHistory (rental_id, car_id, customer_id, rental_date, return_date, rental_cost,
rental_description)
   VALUES (
      RentalHistory_seq.NEXTVAL,
      :new.car_id,
      :new.customer_id,
      :new.start_date,
      :new.end_date,
      :new.total_cost,
      'Car returned'
   );
 END IF;
END;
/
```



Добавлено примечание ([a15]): CREATE OR REPLACE TRIGGER: Это ключевое слово, которое начинает создание триггера. CREATE создает новый триггер, а OR REPLACE позволяет заменить существующий триггер с тем же именем, если он уже существует.

RentalHistoryTrigger: Это имя триггера.

AFTER INSERT ON Orders: Это событие, на которое триггер будет реагировать. В данном случае, триггер сработает после вставки новой записи в таблицу "Orders".

FOR EACH ROW: Это указывает, что триггер будет выполняться для каждой вставленной строки (записи) в таблице "Orders".

BEGIN...END: Здесь начинается тело триггера, где определяется его действие.

INSERT INTO RentalHistory...: В этой части определено, что при каждой вставке в "Orders", триггер создаст новую запись в таблице "RentalHistory" с информацией о сдаче автомобиля в аренду.

RentalHistory_seq.NEXTVAL: Это выражение генерирует уникальное значение для столбца "rental_id" в "RentalHistory".

:new.car_id, :new.customer_id: Это значения, взятые из вставленной строки в таблицу "Orders". Они связываются с соответствующими столбцами в "RentalHistory".

:new.start_date, :new.end_date, :new.total_cost: Аналогично, это данные из вставленной строки в "Orders".

...

Добавлено примечание ([a16]): CREATE OR REPLACE TRIGGER ReturnHistoryTrigger: Эта часть определяет создание нового триггера с именем "ReturnHistoryTrigger" или его замену, если он уже существует с таким именем.

AFTER UPDATE ON Orders: Это определяет событие, при котором триггер будет срабатывать. В данном случае, триггер срабатывает после выполнения операции обновления (UPDATE) в таблице "Orders".

FOR EACH ROW: Это указывает, что триггер выполняется для каждой строки, обновленной в таблице "Orders".

BEGIN ... END;: Здесь располагается основная логика триггера, включая условия и действия, которые будут выполнены.

IF :new.end_date IS NOT NULL AND :old.end_date IS NULL THEN: Это условие, которое проверяет, что новое значение поля "end_date" (:new.end_date) не является NULL, а старое значение (:old.end_date) было NULL. Если это условие выполняется, это означает, что заказ завершается (машина возвращается).

INSERT INTO RentalHistory ...: Это действие триггера. Если условие из предыдущего шага истинно, то выполняется вставка новой записи в таблицу "RentalHistory" с информацией о возвращении машины в аренду.

RentalHistory_seq.NEXTVAL: Это функция, используемая для генерации нового уникального значения для столбца "rental_id".
:new.car_id, :new.customer_id, :new.start_date, :new.end_date, :new.total_cost: Это значения, взятые из обновленной строки в таблице "Orders".

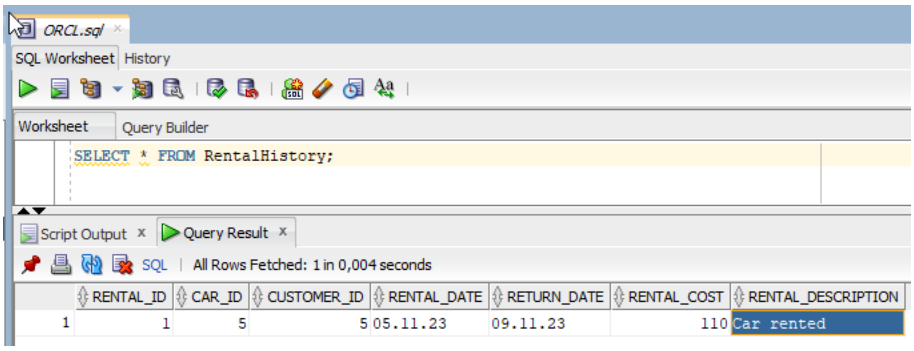
...

Чтобы проверить сохранение истории сдачи в аренду авто, создадим тестовую ситуацию, выполним команду:

```
INSERT INTO Orders (order_id, customer_id, car_id, start_date, end_date, total_cost)
VALUES (6, 5, 5, TO_DATE('2023-11-05', 'YYYY-MM-DD'), TO_DATE('2023-11-09', 'YYYY-MM-DD'),
100.00);
```

Для проверки сохранения истории сдачи в аренду автомобиля выполним SQL-запрос, который вернет записи из таблицы «RentalHistory»:

```
SELECT * FROM RentalHistory;
```



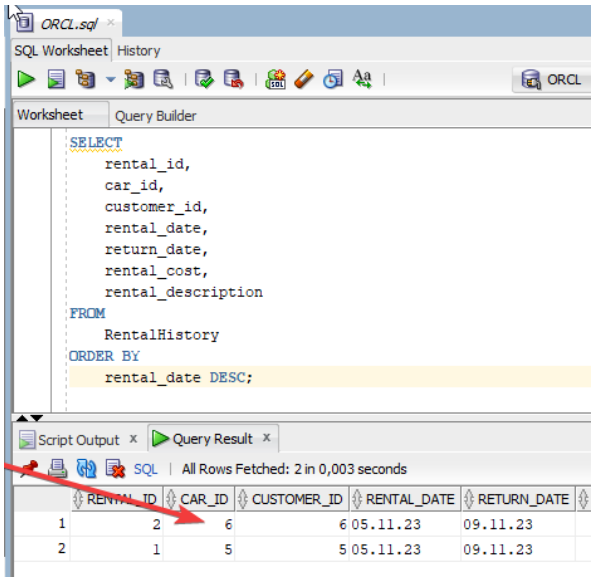
f. отчет об истории сдачи в аренду авто.

Выполним запрос:

```
SELECT
   rental_id,
   car_id,
   customer_id,
   rental_date,
   return_date,
   rental_cost,
   rental_description
FROM
   RentalHistory
ORDER BY
   rental_date DESC;
```

Чтобы проверить работу отчета об истории сдачи в аренду авто добавим информацию об аренде:

```
INSERT INTO Orders (order_id, customer_id, car_id, start_date, end_date, total_cost)
VALUES (7, 6, 6, TO_DATE('2023-11-05', 'YYYY-MM-DD'), TO_DATE('2023-11-08', 'YYYY-MM-DD'),
120.00);
```



Добавлено примечание ([a17]): Этот запрос выбирает следующие данные из таблицы "RentalHistory" и упорядочивает их по дате аренды (сначала новые записи):

rental_id: Уникальный идентификатор аренды.
car_id: Идентификатор автомобиля, который арендован.
customer_id: Идентификатор клиента, который арендовал автомобиль.
rental_date: Дата начала аренды.
return_date: Дата возврата автомобиля.
rental_cost: Стоимость аренды.
rental_description: Описание события аренды.

g. отчет о финансовой деятельности сервиса за период (для простоты будем считать, что стоимость сдачи в аренду на сутки одного автомобиля константна).

Для создания отчета о финансовой деятельности сервиса сдачи автомобилей в аренду за определенный период будем использовать SQL-запрос, который учитывает стоимость аренды на сутки и агрегирует данные из таблицы «RentalHistory».

Подготовка данных:

Прежде чем создавать отчет, должна быть таблица «RentalHistory» с данными о сдаче в аренду автомобилей.

SQL-запрос:

```
SELECT
   TO_CHAR(rental_date, 'YYYY-MM') AS rental_month,
   SUM(rental_cost) AS total_revenue
FROM
   RentalHistory
WHERE
   rental_date >= TO_DATE('2023-11-01', 'YYYY-MM-DD')
   AND rental_date < TO_DATE('2023-12-01', 'YYYY-MM-DD')
GROUP BY
   TO_CHAR(rental_date, 'YYYY-MM')
ORDER BY
   rental_month;
```

Добавлено примечание ([a18]): Данный SQL-запрос выполняет агрегацию данных из таблицы RentalHistory для создания отчета о доходах (в данном случае за один месяц). Отчет будет включать общий доход для каждого месяца.

SELECT TO_CHAR(rental_date, 'YYYY-MM') AS rental_month: В этой части выбираются данные и преобразуется столбец rental_date в формат 'YYYY-MM', что представляет собой год и месяц аренды.

SUM(rental_cost) AS total_revenue: Эта часть запроса выполняет агрегацию данных. Она суммирует стоимость аренды (rental_cost) для каждого месяца и предоставляет результат в столбце total_revenue.

FROM RentalHistory: Указывает источник данных, из которого будут извлекаться записи. В данном случае, это таблица RentalHistory.

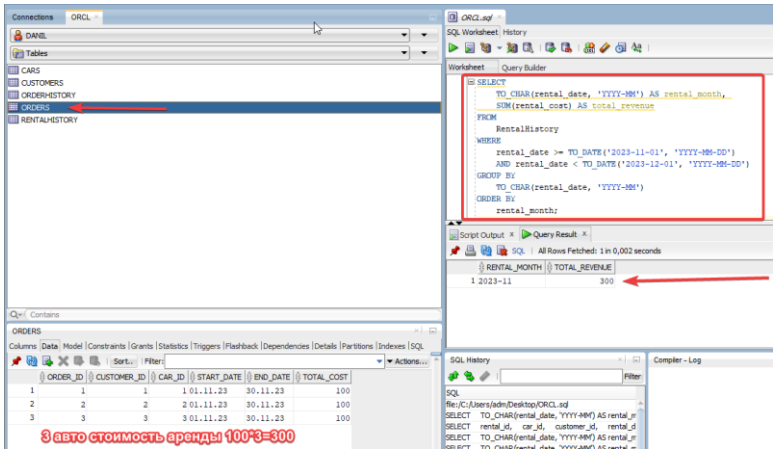
WHERE rental_date >= TO_DATE('2023-01-01', 'YYYY-MM-DD') AND rental_date < TO_DATE('2023-02-01', 'YYYY-MM-DD')

GROUP BY TO_CHAR(rental_date, 'YYYY-MM'): Здесь данные группируются по столбцу rental_date после его преобразования в формат 'YYYY-MM'. Это позволяет сгруппировать данные по месяцам.

ORDER BY rental_month: Записи сортируются в результате по месяцам (в формате 'YYYY-MM') в порядке возрастания.

Для проверки работы SQL-запроса удалю все записи из таблиц Orders, OderHistory, Rentalhistory.
После чего внесу данные о заказах снова:

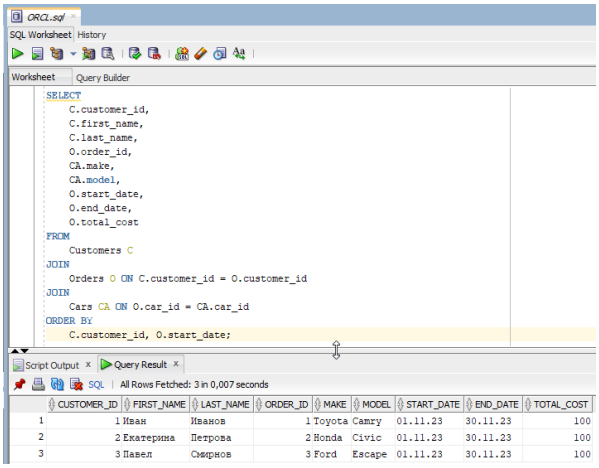
```
INSERT INTO Orders (order_id, customer_id, car_id, start_date, end_date, total_cost)
VALUES (1,1, 1, TO_DATE('2023-11-01', 'YYYY-MM-DD'), TO_DATE('2023-11-30', 'YYYY-MM-DD'),
100.00);

INSERT INTO Orders (order_id, customer_id, car_id, start_date, end_date, total_cost)
VALUES (2,2, 2, TO_DATE('2023-11-01', 'YYYY-MM-DD'), TO_DATE('2023-11-30', 'YYYY-MM-DD'),
100.00);

INSERT INTO Orders (order_id, customer_id, car_id, start_date, end_date, total_cost)
VALUES (3,3, 3, TO_DATE('2023-11-01', 'YYYY-MM-DD'), TO_DATE('2023-11-30', 'YYYY-MM-DD'),
100.00);
```



h. Отчет о клиентах: когда какие авто брал в аренду, сколько заплатил и т.д.

Для создания отчета о клиентах, их заказах, арендованных автомобилях и общих расходах мы будем использовать SQL-запрос:

```
SELECT
    C.customer_id,
    C.first_name,
    C.last_name,
    O.order_id,
    CA.make,
    CA.model,
    O.start_date,
    O.end_date,
    O.total_cost
FROM
    Customers C
JOIN
    Orders O ON C.customer_id = O.customer_id
JOIN
    Cars CA ON O.car_id = CA.car_id
ORDER BY
    C.customer_id, O.start_date;
```



Добавлено примечание ([a19]): SELECT - начинаем запрос и выбираем столбцы, которые будут включены в результат:

C.customer_id - идентификатор клиента.
C.first_name - имя клиента.
C.last_name - фамилия клиента.
O.order_id - идентификатор заказа.
CA.make - марка автомобиля.
CA.model - модель автомобиля.
O.start_date - дата начала аренды.
O.end_date - дата окончания аренды.
O.total_cost - общая стоимость заказа.
FROM - указываем таблицы, из которых будут выбраны данные:

Customers C - таблица клиентов.
Orders O - таблица заказов.
Cars CA - таблица автомобилей.
JOIN - объединяем таблицы на основе связей между ними:

ON C.customer_id = O.customer_id - связываем таблицы Customers и Orders по идентификатору клиента.
ON O.car_id = CA.car_id - связываем таблицы Orders и Cars по идентификатору арендованного автомобиля.
ORDER BY - задаем порядок сортировки результатов:

C.customer_id - сортируем результаты по идентификатору клиента.
O.start_date - затем по дате начала аренды.

i. Отчет о текущем состоянии автопарка: сколько в аренде, сколько свободны, сколько на ТО и т.д.

Для создания отчета о клиентах, их заказах, арендованных автомобилях и общих расходах мы будем использовать SQL-запрос:

```
SELECT
   COUNT(CASE WHEN status = 'в аренде' THEN 1 ELSE NULL END) AS cars_in_rent,
   COUNT(CASE WHEN status = 'свободен' THEN 1 ELSE NULL END) AS cars_available,
   COUNT(CASE WHEN status = 'на ТО' THEN 1 ELSE NULL END) AS cars_under_maintenance
FROM Cars;
```

Добавлено примечание ([a20]): В этом запросе мы используем функцию COUNT с условными выражениями CASE WHEN, чтобы подсчитать количество автомобилей в каждой категории (в аренде, свободных, на ТО). Результат запроса будет содержать три столбца: "cars_in_rent" (автомобили в аренде), "cars_available" (свободные автомобили) и "cars_under_maintenance" (автомобили на ТО).

На данном этапе была установлена проблема, что после добавления информации в таблицу Orders, у нас не обновляется информация в таблице Cars (не меняется статус), для обновления мы будем использовать триггер:

```
CREATE OR REPLACE TRIGGER UpdateCarStatusTrigger
AFTER INSERT OR UPDATE OR DELETE ON Orders
FOR EACH ROW
BEGIN
 IF INSERTING THEN
   UPDATE Cars
   SET status = 'в аренде'
   WHERE car_id = :new.car_id;
 ELSIF DELETING THEN
   UPDATE Cars
   SET status = 'доступен'
   WHERE car_id = :old.car_id;
 END IF;
END;
/
```

Добавлено примечание ([a21]): Данный SQL скрипт создает триггер с именем "UpdateCarStatusTrigger", который автоматически обновляет статус автомобиля в таблице "Cars" при вставке, обновлении или удалении данных в таблице "Orders".

CREATE OR REPLACE TRIGGER UpdateCarStatusTrigger: Это начало создания триггера с именем "UpdateCarStatusTrigger".

AFTER INSERT OR UPDATE OR DELETE ON Orders: Триггер срабатывает после операций вставки, обновления и удаления в таблице "Orders".

FOR EACH ROW: Этот триггер будет выполняться для каждой строки, которая затрагивается операцией INSERT, UPDATE или DELETE.

BEGIN: Начало тела триггера.

IF INSERTING THEN: Эта часть проверяет, срабатывает ли триггер после операции вставки (INSERT). Если условие выполняется, выполняются следующие действия:

UPDATE Cars SET status = 'в аренде' WHERE car_id = :new.car_id;: Здесь выполняется SQL-запрос для обновления статуса автомобиля в таблице "Cars" на "в аренде" для автомобиля, связанного с данным заказом (определенным как :new.car_id).

ELSIF DELETING THEN: Если триггер срабатывает после операции удаления (DELETE), выполняются следующие действия:

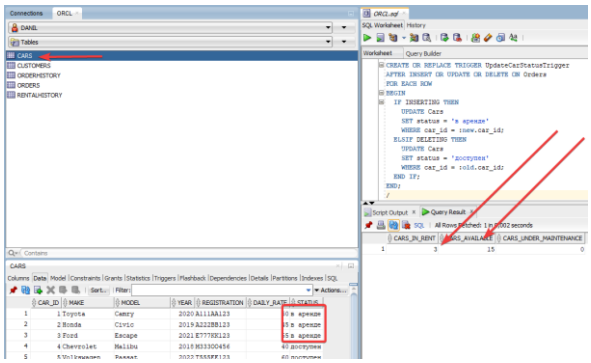
UPDATE Cars SET status = 'свободен' WHERE car_id = :old.car_id;: Здесь выполняется SQL-запрос для обновления статуса автомобиля в таблице "Cars" на "доступен" для автомобиля, который был арендован в заказе, который удаляется (определенный как :old.car_id).

END IF;: Завершение условной конструкции.

END;: Завершение тела триггера.

Для проверки работы SQL-запроса удалю все записи из таблиц Orders, OderHistory, Rentalhistory.
После чего внесу данные о заказах снова:

```
INSERT INTO Orders (order_id, customer_id, car_id, start_date, end_date, total_cost)
VALUES (1,1, 1, TO_DATE('2023-11-01', 'YYYY-MM-DD'), TO_DATE('2023-11-30', 'YYYY-MM-DD'),
100.00);

INSERT INTO Orders (order_id, customer_id, car_id, start_date, end_date, total_cost)
VALUES (2,2, 2, TO_DATE('2023-11-01', 'YYYY-MM-DD'), TO_DATE('2023-11-30', 'YYYY-MM-DD'),
100.00);

INSERT INTO Orders (order_id, customer_id, car_id, start_date, end_date, total_cost)
VALUES (3,3, 3, TO_DATE('2023-11-01', 'YYYY-MM-DD'), TO_DATE('2023-11-30', 'YYYY-MM-DD'),
100.00);
```



j. Подбор(поиск) авто по параметрам

Для реализации поиска автомобилей по параметрам воспользуемся SQL-запросами и WHERE-условиями, чтобы фильтровать данные в таблице "Cars" в соответствии с заданными параметрами:

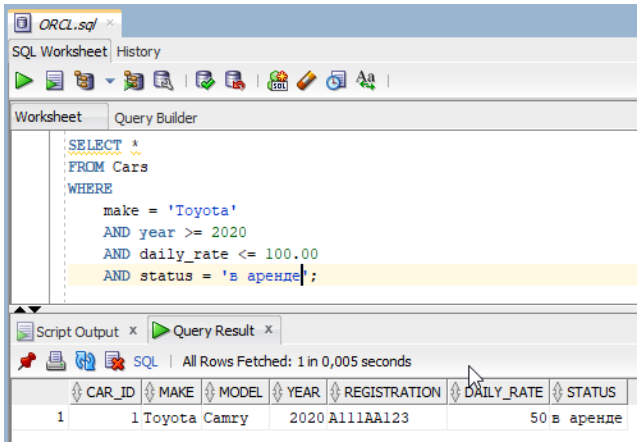
```
SELECT *
FROM Cars
WHERE
   make = 'Toyota'
   AND year >= 2020
   AND daily_rate <= 100.00
   AND status = 'в аренде';
```

Добавлено примечание ([a22]): SELECT * выбирает все столбцы из таблицы "Cars". Мы можем указать только те столбцы, которые нас интересуют, вместо *, если нам не нужна вся информация.

FROM Cars указывает таблицу, из которой нужно извлечь данные.

WHERE начинает блок условий для фильтрации результатов.

Каждая строка в блоке WHERE представляет собой условие для фильтрации. В приведенном примере показаны условия фильтрации по марке, году выпуска, дневной ставке аренды и статусу.




```
SELECT *
FROM Cars
WHERE
   make = 'BMW'
   AND year >= 2021
   AND daily_rate <= 100.00
   AND status = 'доступен';
```