Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»
Факультет инфокоммуникационных технологий

ОТЧЕТ
О ЛАБОРАТОРНОЙ РАБОТЕ № 3

по теме:

«Процедуры, функции, триггеры в PostgreSQL»

по дисциплине: Проектирование и реализация баз данных

Специальность:
45.03.04 Интеллектуальные системы в гуманитарной сфере

Проверила:
Говорова М.М.
Дата: «..» … 2023 г.
Оценка _____

Выполнил:
студент группы К32422
Малаев С.Г.

Санкт-Петербург 2022/2023

Цель работы: овладеть практическими создания и использования процедур, функций и триггеров в базе данных PostgreSQL.

Практическое задание:

Вариант 1

1. Создать процедуры/функции согласно индивидуальному заданию и (согласно индивидуальному заданию, часть 4).

2. Создать триггер для логирования событий вставки, удаления, редактирования данных в базе данных PostgreSQL (согласно индивидуальному заданию, часть 5). Допустимо создать универсальный триггер или отдельные триггеры на логирование действий.

Вариант 2

1. Создать процедуры/функции согласно индивидуальному заданию и (согласно индивидуальному заданию, часть 4).
2.
    2.1. Модифицировать триггер (триггерную функцию) на проверку корректности входа и выхода сотрудника (см. Практическое задание 1 Лабораторного практикума (Приложение)) с максимальным учетом «узких» мест некорректных данных по входу и выходу.
    2.2. Создать авторский триггер по варианту индивидуального задания.

Предметная область: Автомастерская (вариант 11)

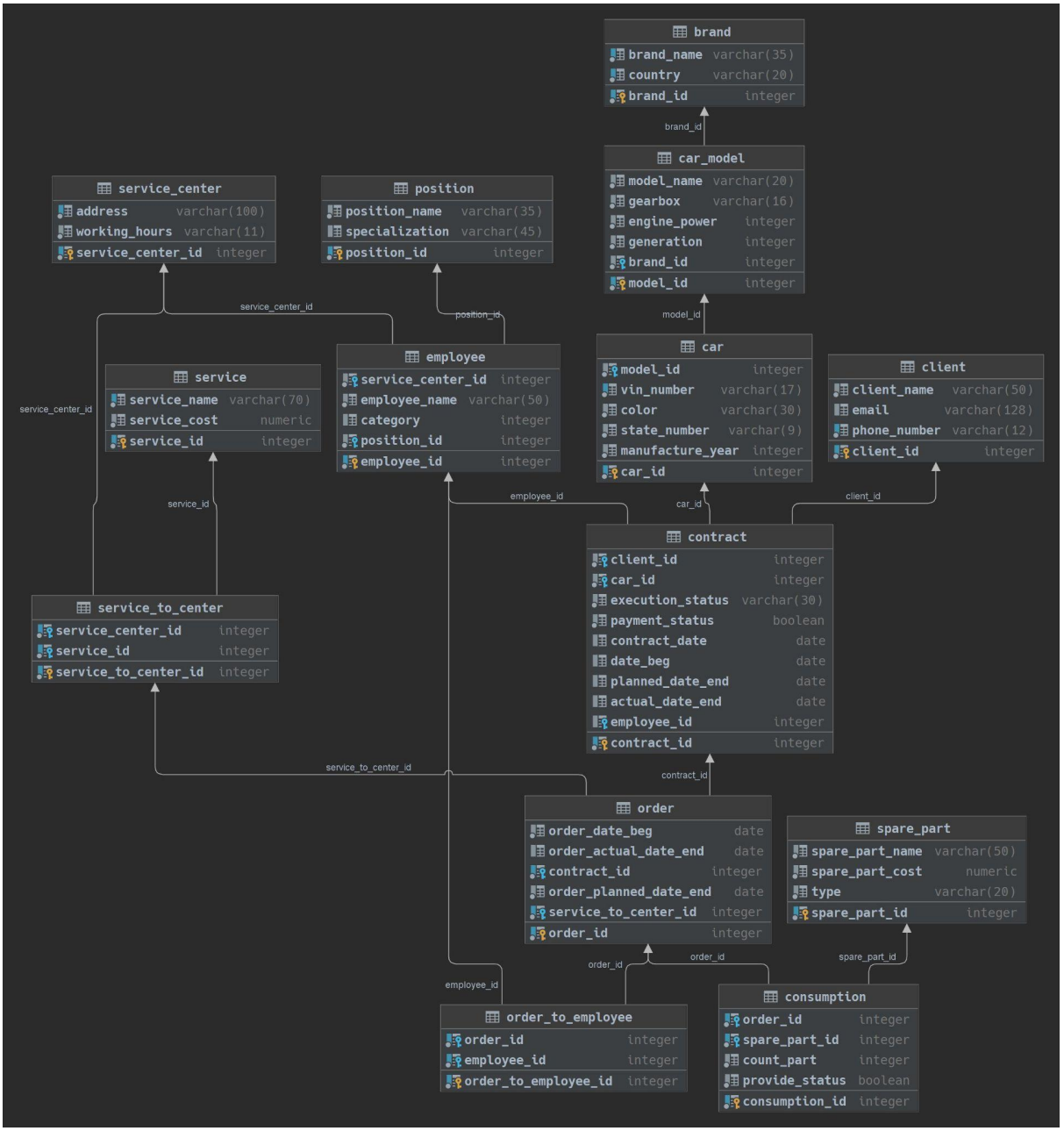

Рисунок 1 – ERD базы данных

Выполнение работы:

Индивидуальное задание:

Задание 4. Создать хранимые процедуры:

1. Повышения цены деталей для автомобиля “Ford” на 10 %.
2. Для повышения разряда тех мастеров, которые отремонтировали больше 3 автомобилей.
3. Сколько автомобилей отремонтировал каждый механик за истекший квартал.

Выполнение:

1. Повышения цены деталей для автомобиля “Ford” на 10 %.

   Изменения: Так как детали не привязаны к автомобилям, я создал процедуру которая изменяет все цены деталей определенного типа на основе входных параметров (*percentage* DECIMAL, *type_name* VARCHAR), где *percentage* коэффициент изменения цены, а *type_name* название типа детали.

   Данные до изменения:

	spare_part_id	spare_part_name	spare_part_cost	type
1	1	Блок управления ABS	7600	Электроника
2	2	Генератор	9900	Электроника
3	3	Шаровая	2000	Ходовая
4	4	Тормозной суппорт	1000	Электроника
5	5	Тормозные диски	2500	Ходовая
6	6	Датчик распредвала	400	Электроника
7	7	Поперечный рычаг	1200	Навесное
8	8	Сальники коленвала	500	Расходники
9	9	Карданный вал	5500	Ходовая
10	10	Прокладка выпускного коллектора	250	Расходники

Создание процедуры:

```
CREATE OR REPLACE PROCEDURE
raise_electrical_spare_part_cost(percentage DECIMAL, type_name
VARCHAR)
LANGUAGE plpgsql
AS $$
BEGIN
    UPDATE spare_part
    SET spare_part_cost = spare_part_cost * percentage
    WHERE type = type_name;
END;
$$;
```

Данные после изменения:

	spare_part_id	spare_part_name	spare_part_cost	type
1	1	Блок управления ABS	8360	Электроника
2	2	Генератор	10890	Электроника
3	3	Шаровая	2000	Ходовая
4	4	Тормозной суппорт	1100	Электроника
5	5	Тормозные диски	2500	Ходовая
6	6	Датчик распредвала	440	Электроника
7	7	Поперечный рычаг	1200	Навесное
8	8	Сальники коленвала	500	Расходники
9	9	Карданный вал	5500	Ходовая
10	10	Прокладка выпускного коллектора	250	Расходники

2. Для повышения разряда тех мастеров, которые отремонтировали больше 3 автомобилей.

Данные до изменения:

	employee_id	service_center_id	employee_name	category	position_id
1	1	1	Широкова Софья Александровна	<null>	17
2	2	1	Савельев Степан Максимович	<null>	11
3	3	1	Смирнова София Тимофеевна	<null>	41
4	4	1	Петров Глеб Игоревич	2	18
5	5	1	Тарасова Варвара Игоревна	3	36
6	6	1	Дмитриева Кира Платоновна	5	9
7	7	1	Горбунова Сафия Степановна	<null>	16
8	8	1	Павлов Александр Даниилович	9	21
9	9	1	Дмитриев Даниил Янович	5	4
10	10	1	Орлов Юрий Фёдорович	7	15

Создание процедуры:

```
CREATE OR REPLACE PROCEDURE raise_employee_category()
LANGUAGE plpgsql
AS $$
BEGIN
    UPDATE employee
    SET category = LEAST(category + 1, 9)
    WHERE employee_id IN (
        SELECT employee_id
        FROM order_to_employee
            JOIN "order" USING(order_id)
            JOIN contract USING(contract_id)
        GROUP BY employee_id
        HAVING COUNT(DISTINCT car_id) > 3
    );
END;
$$;
```

Данные после изменения:

	employee_id	service_center_id	employee_name	category	position_id
1	1	1	Широкова Софья Александровна	<null>	17
2	2	1	Савельев Степан Максимович	<null>	11
3	3	1	Смирнова София Тимофеевна	<null>	41
4	4	1	Петров Глеб Игоревич	3	18
5	5	1	Тарасова Варвара Игоревна	4	36
6	6	1	Дмитриева Кира Платоновна	6	9
7	7	1	Горбунова Сафия Степановна	<null>	16
8	8	1	Павлов Александр Даниилович	9	21
9	9	1	Дмитриев Даниил Янович	6	4
10	10	1	Орлов Юрий Фёдорович	7	15

3. Сколько автомобилей отремонтировал каждый механик за истекший квартал.

   Создание функции:

```
CREATE OR REPLACE FUNCTION employee_cars()
RETURNS TABLE(employee_id INT, employee_name VARCHAR, car_count
BIGINT)
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN QUERY
    SELECT
        e.employee_id, e.employee_name,
        COUNT(car_id) AS car_count
    FROM employee e
        JOIN order_to_employee USING(employee_id)
        JOIN "order" USING(order_id)
        JOIN contract USING(contract_id)
    GROUP BY e.employee_id;
END;
$$;
```

   Возвращает:

	employee_id	employee_name	car_count 1
1	32	Емельянов Иван Егорович	133
2	103	Романова Василиса Максимовна	131
3	71	Афанасьев Денис Андреевич	130
4	8	Павлов Александр Даниилович	127
5	33	Семенов Илья Александрович	125
6	52	Попов Никита Тимофеевич	124
7	45	Зиновьев Павел Кириллович	124
8	82	Королев Степан Арсентьевич	123
9	48	Коротков Кирилл Александрович	123
10	96	Волошина Василиса Степановна	123

4. Создать триггер для логирования событий вставки, удаления, редактирования данных.

Описание: триггер регистрирует каждую операцию INSERT, UPDATE и DELETE, выполненную над таблицей *“order”* в отдельную таблицу *“order_audit_log”*. Этот журнал может помочь отслеживать изменения, контролировать производительность работников и анализировать тенденции на заказы.

Для начала необходимо создать журнал логирования:

```
CREATE TABLE order_audit_log (
    id SERIAL PRIMARY KEY,
    operation VARCHAR(10) NOT NULL,
    order_id INTEGER,
    order_date_beg DATE,
    order_actual_date_end DATE,
    contract_id INTEGER,
    order_planned_date_end DATE,
    service_to_center_id INTEGER,
    log_timestamp TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);
```

Затем создадим триггерную функцию для регистрации изменений:

```
create function log_order_changes() returns trigger
    language plpgsql
as
$$
BEGIN
    IF TG_OP = 'INSERT' THEN
        INSERT INTO order_audit_log (operation, order_id, order_date_beg, order_actual_date_end, contract_id,
                                     order_planned_date_end, service_to_center_id)
        VALUES ('INSERT', NEW.order_id, NEW.order_date_beg, NEW.order_actual_date_end, NEW.contract_id,
                NEW.order_planned_date_end, NEW.service_to_center_id);
        RETURN NEW;
    ELSIF TG_OP = 'UPDATE' THEN
        INSERT INTO order_audit_log (operation, order_id, order_date_beg, order_actual_date_end, contract_id,
                                     order_planned_date_end, service_to_center_id)
        VALUES ('UPDATE', NEW.order_id, NEW.order_date_beg, NEW.order_actual_date_end, NEW.contract_id,
                NEW.order_planned_date_end, NEW.service_to_center_id);
        RETURN NEW;
    ELSIF TG_OP = 'DELETE' THEN
        INSERT INTO order_audit_log (operation, order_id, order_date_beg, order_actual_date_end, contract_id,
                                     order_planned_date_end, service_to_center_id)
        VALUES ('DELETE', OLD.order_id, OLD.order_date_beg, OLD.order_actual_date_end, OLD.contract_id,
                OLD.order_planned_date_end, OLD.service_to_center_id);
        RETURN OLD;
    END IF;
END;
```

И наконец, создадим триггеры, которые будут выполнять функцию после операций INSERT, UPDATE и DELETE в таблице *“order”*:

```
CREATE TRIGGER log_order_changes_after_insert
AFTER INSERT OR UPDATE OR DELETE ON "order"
FOR EACH ROW
EXECUTE FUNCTION log_order_changes();
```

Проверим работу нашего триггера:

```
BEGIN;

INSERT INTO "order"(order_date_beg, contract_id,
order_planned_date_end, service_to_center_id)
SELECT '2023-05-11', contract_id, '2023-05-18',
service_to_center_id
FROM contract, service_to_center
WHERE contract_id = 1
    AND service_to_center_id = 1;

DELETE FROM "order"
WHERE order_id = (SELECT MAX(order_id) FROM "order");


UPDATE "order"
SET order_actual_date_end = '2023-05-17', order_planned_date_end =
'2023-05-17'
WHERE order_id = 1;

SELECT * FROM order_audit_log;

COMMIT;
```

Результат:

#	id	operation	order_id	order_date_beg	order_actual_date_end	contract_id	order_planned_date_end	service_to_center_id	log_timestamp
1	1	INSERT	3904	2023-05-11		1	2023-05-18	1	2023-05-11 02:28:07.220828
2	2	DELETE	3904	2023-05-11		1	2023-05-18	1	2023-05-11 02:28:07.220828
3	3	UPDATE	1	2021-02-24	2023-05-17	5	2023-05-17	218	2023-05-11 02:28:07.220828

Заключение:

В ходе выполнения данной работы, были изучены и реализованы различные функции и процедуры в PostgreSQL. Кроме того, был создан триггер логирования на операции INSERT, UPDATE и DELETE.