Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

ОТЧЕТ

по Лабораторной работе № 5

«процедуры, функции, триггеры в PostgreSQL»

по дисциплине «Проектирование и реализация баз данных»

Обучающийся Ермаков Максим Олегович
Факультет прикладной информатики
Группа K3240
Направление подготовки 09.03.03 Прикладная информатика
Образовательная программа Мобильные и сетевые технологии 2023
Преподаватель Говорова Марина Михайловна

Санкт-Петербург
2025

СОДЕРЖАНИЕ

Оглавление

ВВЕДЕНИЕ

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

Практическое задание (min - 6 баллов, max - 10 баллов, доп. баллы - 3):

1. Создать 3 процедуры для индивидуальной БД согласно варианту (часть 4 ЛР2). Допустимо использование IN/OUT параметров. Допустимо создать авторские процедуры. (3 балла)

2. Создать триггеры для индивидуальной БД согласно варианту:

Вариант 2.1. 3 триггера - 3 балла (min). Допустимо использовать триггеры логирования из практического занятия по функциям и триггерам.

Вариант 2.2. 7 оригинальных триггеров - 7 баллов (max).

1 Выполнение

1.1 Название создаваемой базы данных

БД «Оптовая база»

Описание предметной области: Оптовая база закупает товары у компаний-поставщиков и поставляет их компаниям – покупателям. Компании поставщики не являются производителями товара. Доход оптовой базы составляет не менее 5% от стоимости товара, проданного компании-покупателю. Каждый товар имеет производителя. Один и тот же товар может доставляться несколькими поставщиками, и один и тот же поставщик может поставлять несколько видов товаров. Цены поставки товара у разных поставщиков могут отличаться. В один заказ при покупке товара у оптовой базы может попасть товар от разных поставщиков, в зависимости от наличия на складе. Поставки и заказы обслуживают менеджеры по работе с клиентами (по поставкам и продажам).

БД должна содержать следующий минимальный набор сведений: Табельный номер. Код сотрудника. Паспортные данные сотрудника. Должность. Код товара. Название товара. Единица измерения товара. Количество товара. Запас товара на базе. Стоимость единицы товара. Код поставки. Дата поставки на базу. Количество поставки. Примечание – описание товара. Код поставщика. Название компании поставщика. Адрес

поставщика. Дата поставки. Количество товара в партии. Номер счета. Код организации – покупателя. Название компании покупателя. Адрес покупателя. Дата заказа. Дата вывоза. Номер партии. Продажная цена товара. Должность сотрудника. Количество ставок (по штатному расписанию).

1.2 Схема логической модели базы данных, сгенерированная в Generate ERD

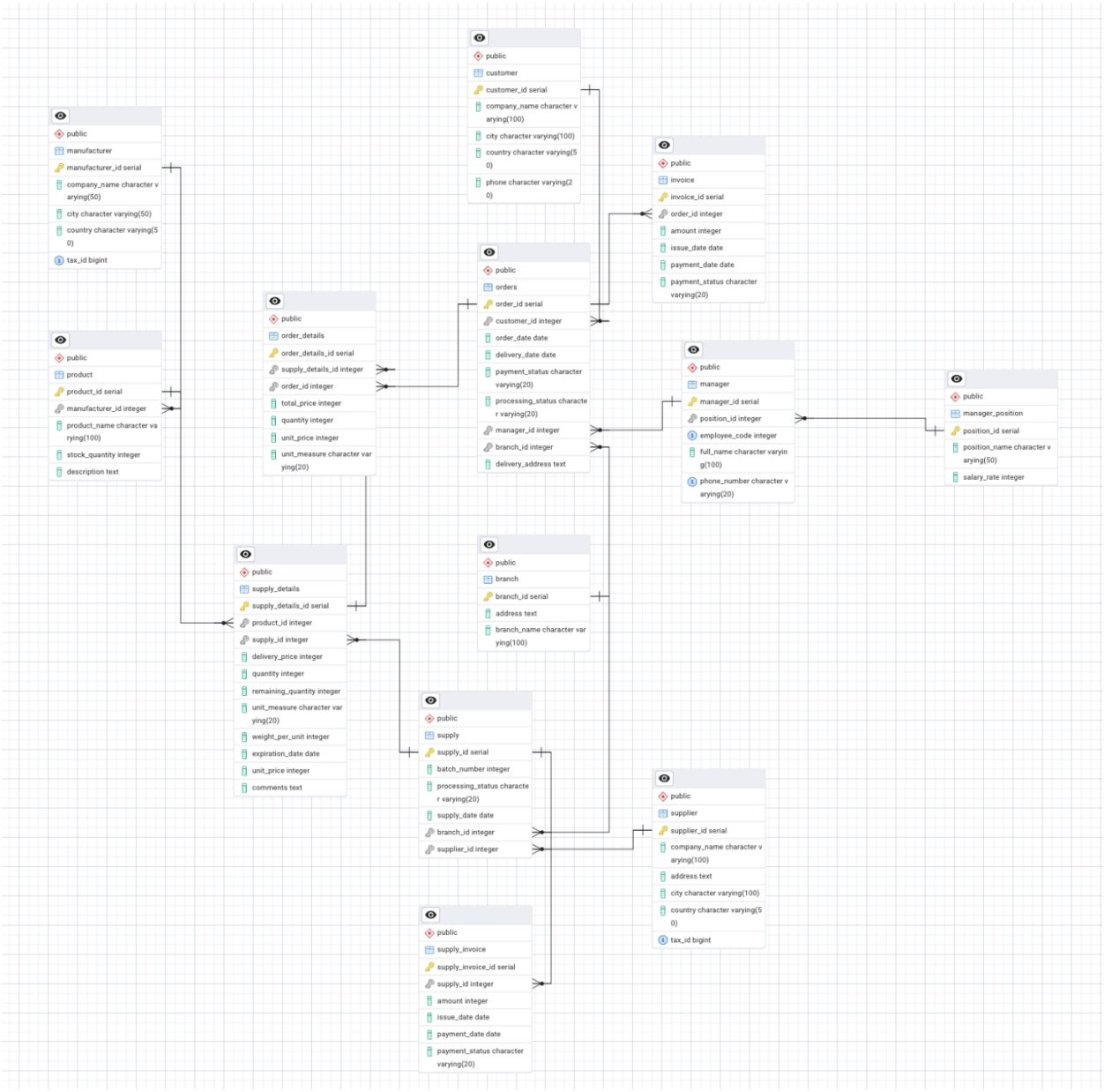


Рисунок 1 – Схема логической модели базы данных, сгенерированная в Generate ERD

1.3 Создание хранимых процедур

1.3.1 Процедура уменьшения цены на заданный процент у товаров, которые хранятся дольше допустимого срока

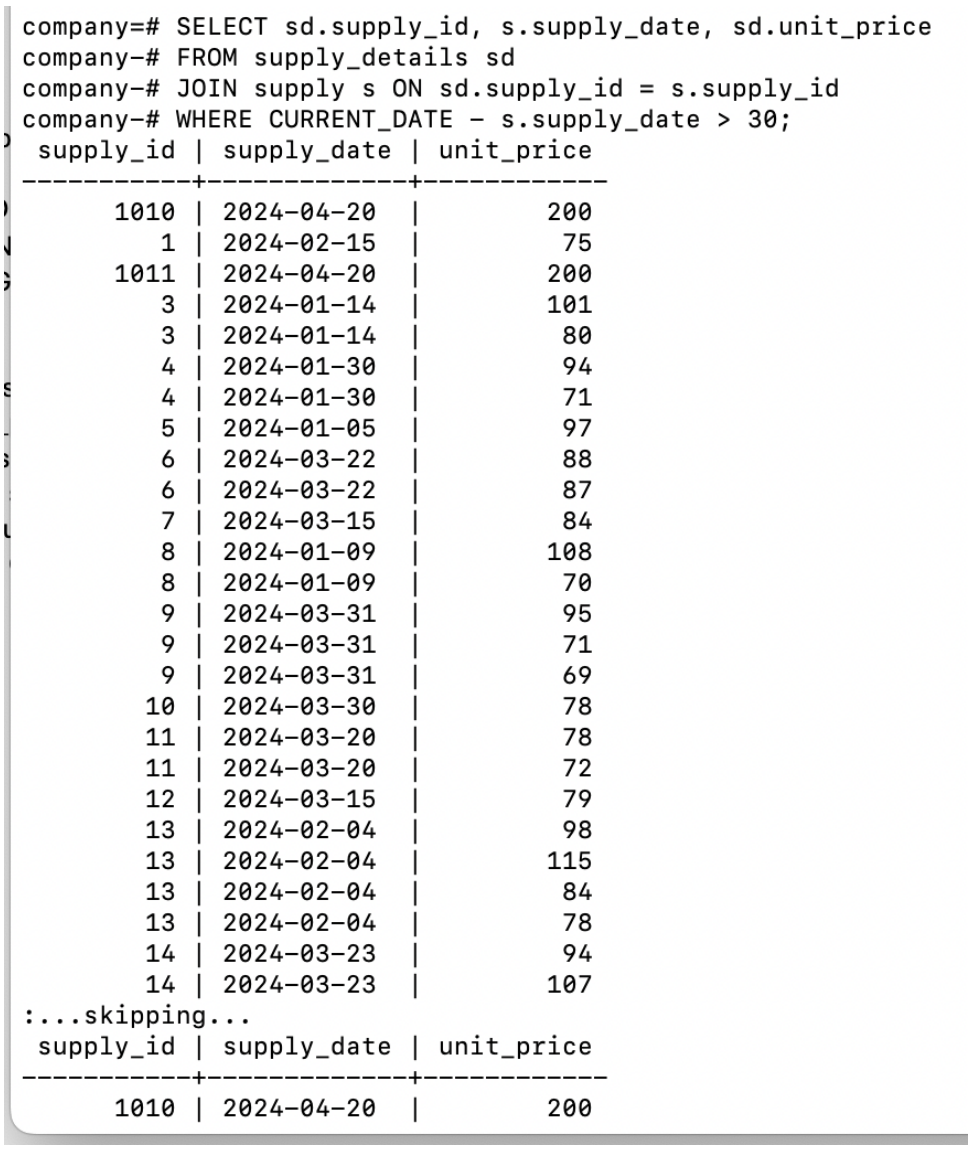
Выбираем все строки из supply_details, где текущая дата – дата поставки > norm_days
Для этих строк понижаем unit_price

```
CREATE OR REPLACE PROCEDURE reduce_price_by_storage_period(norm_days INT, percent INT)
LANGUAGE plpgsql
AS $$
BEGIN
 UPDATE supply_details
 SET unit_price = unit_price * (1 - percent / 100.0)
 WHERE supply_id IN (
  SELECT supply_id
  FROM supply
  WHERE CURRENT_DATE - supply_date > norm_days
 );
END;
                                                    $$;
```

```
company=# CREATE OR REPLACE PROCEDURE reduce_price_by_storage_period(norm_days INT, percent INT)
company-# LANGUAGE plpgsql
company-# AS $$
company$# BEGIN
company$#   UPDATE supply_details
company$#   SET unit_price = unit_price * (1 - percent / 100.0)
company$#   WHERE supply_id IN (
company$#     SELECT supply_id
company$#     FROM supply
company$#     WHERE CURRENT_DATE - supply_date > norm_days
company$#   );
company$# END;
company$# $$;
CREATE PROCEDURE
```

Рисунок 1 – Скриншот создания процедуры 1

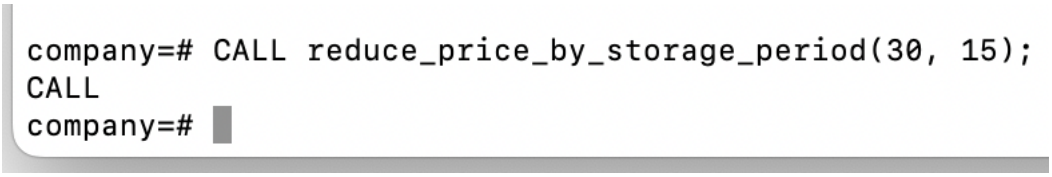
```
SELECT sd.supply_id, s.supply_date, sd.unit_price
FROM supply_details sd
JOIN supply s ON sd.supply_id = s.supply_id
WHERE CURRENT_DATE - s.supply_date > 30;
```



```
company=# SELECT sd.supply_id, s.supply_date, sd.unit_price
company-# FROM supply_details sd
company-# JOIN supply s ON sd.supply_id = s.supply_id
company-# WHERE CURRENT_DATE - s.supply_date > 30;
 supply_id | supply_date | unit_price
-----------+-------------+------------
      1010 | 2024-04-20  |        200
         1 | 2024-02-15  |         75
      1011 | 2024-04-20  |        200
         3 | 2024-01-14  |        101
         3 | 2024-01-14  |         80
         4 | 2024-01-30  |         94
         4 | 2024-01-30  |         71
         5 | 2024-01-05  |         97
         6 | 2024-03-22  |         88
         6 | 2024-03-22  |         87
         7 | 2024-03-15  |         84
         8 | 2024-01-09  |        108
         8 | 2024-01-09  |         70
         9 | 2024-03-31  |         95
         9 | 2024-03-31  |         71
         9 | 2024-03-31  |         69
        10 | 2024-03-30  |         78
        11 | 2024-03-20  |         78
        11 | 2024-03-20  |         72
        12 | 2024-03-15  |         79
        13 | 2024-02-04  |         98
        13 | 2024-02-04  |        115
        13 | 2024-02-04  |         84
        13 | 2024-02-04  |         78
        14 | 2024-03-23  |         94
        14 | 2024-03-23  |        107
:...skipping...
 supply_id | supply_date | unit_price
-----------+-------------+------------
      1010 | 2024-04-20  |        200
```

Рисунок 2 – Цены до применения процедуры

CALL reduce_price_by_storage_period(30, 15);



```
company=# CALL reduce_price_by_storage_period(30, 15);
CALL
company=#
```

Рисунок 3 – Вызов процедуры 2

SELECT sd.supply_id, s.supply_date, sd.unit_price
FROM supply_details sd
JOIN supply s ON sd.supply_id = s.supply_id
WHERE CURRENT_DATE - s.supply_date > 30;

```
company=# SELECT sd.supply_id, s.supply_date, sd.unit_price
FROM supply_details sd
JOIN supply s ON sd.supply_id = s.supply_id
WHERE CURRENT_DATE - s.supply_date > 30;
 supply_id | supply_date | unit_price
-----------+-------------+------------
      1010 | 2024-04-20  |        170
         1 | 2024-02-15  |         64
      1011 | 2024-04-20  |        170
         3 | 2024-01-14  |         86
         3 | 2024-01-14  |         68
         4 | 2024-01-30  |         80
         4 | 2024-01-30  |         60
         5 | 2024-01-05  |         82
         6 | 2024-03-22  |         75
         6 | 2024-03-22  |         74
         7 | 2024-03-15  |         71
         8 | 2024-01-09  |         92
         8 | 2024-01-09  |         60
         9 | 2024-03-31  |         81
         9 | 2024-03-31  |         60
         9 | 2024-03-31  |         59
        10 | 2024-03-30  |         66
        11 | 2024-03-20  |         66
        11 | 2024-03-20  |         61
        12 | 2024-03-15  |         67
        13 | 2024-02-04  |         83
        13 | 2024-02-04  |         98
        13 | 2024-02-04  |         71
        13 | 2024-02-04  |         66
        14 | 2024-03-23  |         80
        14 | 2024-03-23  |         91
        15 | 2024-01-26  |         58
        15 | 2024-01-26  |         79
        15 | 2024-01-26  |         94
        17 | 2024-02-23  |         68
        17 | 2024-02-23  |         65
```

Рисунок 4 – результат работы процедуры

1.3.2 Процедура расчета общей суммы продаж за прошедшие сутки

Смотрим таблицы orders и order_details
Фильтруем заказы с order_date = CURRENT_DATE - 1
Суммируем total_price по этим заказам

```
CREATE OR REPLACE PROCEDURE calculate_daily_income(OUT total INTEGER)
LANGUAGE plpgsql
AS $$
BEGIN
 SELECT COALESCE(SUM(od.total_price), 0)
 INTO total
 FROM orders o
 JOIN order_details od ON o.order_id = od.order_id
 WHERE o.order_date = DATE '2024-03-30';
END;
```

$$;

```
company=# CREATE OR REPLACE PROCEDURE calculate_daily_income(OUT total INTEGER)
company-# LANGUAGE plpgsql
company-# AS $$
company$# BEGIN
company$#   SELECT COALESCE(SUM(od.total_price), 0)
company$#   INTO total
company$#   FROM orders o
company$#   JOIN order_details od ON o.order_id = od.order_id
company$#   WHERE o.order_date = CURRENT_DATE - 1;
company$# END;
company$# $$;
CREATE PROCEDURE
company=#
```

Рисунок 5 – Скриншот создания процедуры 2

CALL calculate_daily_income(total := NULL);

```
company=# CALL calculate_daily_income(total := NULL);
 total
--------
 615200
(1 row)
```

Рисунок 6 – Результат работы процедуры 2

1.3.3 Процедура добавления нового поставщика, если в базе еще нет такого ИНН (tax_id)

- Принимаем параметры нового поставщика: название, город, страну, ИНН.
- Проверяем, есть ли уже поставщик с таким ИНН.
- Если **нет** — добавляем.
- Если **да** — ничего не делаем

CREATE OR REPLACE PROCEDURE add_supplier_if_not_exists(

```
  name VARCHAR,
  city VARCHAR,
  country VARCHAR,
  taxid BIGINT
)
LANGUAGE plpgsql
AS $$
BEGIN
 IF NOT EXISTS (
  SELECT 1 FROM supplier WHERE tax_id = taxid
 ) THEN
  INSERT INTO supplier (company_name, city, country, tax_id)
  VALUES (name, city, country, taxid);
 END IF;
END;
                                                    $$;
```

```
company=# CALL add_supplier_if_not_exists('ООО "Новый поставщик"', 'Казань', 'Россия', 88005553535);
CALL
company=# SELECT * FROM supplier WHERE tax_id = 88005553535;
 supplier_id |     company_name      | address |  city  | country |   tax_id
-------------+-----------------------+---------+--------+---------+-------------
          55 | ООО "Новый поставщик" |         | Казань | Россия  | 88005553535
(1 row)

company=#
```

Рисунок 7 – результат работы процедуры 3

1.4 Создание триггеров

1.4.1 Триггер для логирования новых заказов

Создание таблицы логов

```
company=# CREATE TABLE order_log (
company(#     log_id SERIAL PRIMARY KEY,
company(#     order_id INTEGER,
company(#     customer_id INTEGER,
company(#     created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
company(# );
CREATE TABLE
company=#
```

Рисунок 8 – создание таблицы логов

функция для триггера

```
company=# CREATE OR REPLACE FUNCTION log_order_insert()
company-# RETURNS TRIGGER AS $$
company$# BEGIN
company$#   INSERT INTO order_log(order_id, customer_id)
company$#   VALUES (NEW.order_id, NEW.customer_id);
company$#   RETURN NEW;
company$# END;
company$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
company=#
```

Рисунок 9 – создание функции для триггера

сам триггер

```
company=# CREATE TRIGGER trg_log_order_insert
company-# AFTER INSERT ON orders
company-# FOR EACH ROW
company-# EXECUTE FUNCTION log_order_insert();
CREATE TRIGGER
company=#
```

Рисунок 10 – Создание триггера

```
company=# INSERT INTO orders (customer_id, order_date, delivery_date, payment_status, processing_status, ma
nager_id, branch_id)
company-# VALUES (500, '2024-04-27', '2024-04-30', 'ожидание', 'новый', 1, 1);
INSERT 0 1
company=# SELECT * FROM order_log ORDER BY log_id DESC;
 log_id | order_id | customer_id |         created_at
--------+----------+-------------+----------------------------
      1 |     1002 |         500 | 2025-05-04 18:24:30.303133
(1 row)

company=#
```

Рисунок 11 – Тестирование работы триггера

1.4.2 Триггер логирования удаления продуктов

Создание таблицы логов удаления:

```
CREATE TABLE product_delete_log (
   log_id SERIAL PRIMARY KEY,
   product_id INTEGER,
   product_name VARCHAR(100),
   deleted_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
                                                );
```

функция триггера

```
company=# CREATE OR REPLACE FUNCTION log_product_delete()
company-# RETURNS TRIGGER AS $$
company$# BEGIN
company$#   INSERT INTO product_delete_log(product_id, product_name)
company$#   VALUES (OLD.product_id, OLD.product_name);
company$#   RETURN OLD;
company$# END;
company$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
company=#
```

Рисунок 12 – Создание функции триггера

создание триггера

```
company=# CREATE TRIGGER trg_log_product_delete
company-# AFTER DELETE ON product
company-# FOR EACH ROW
company-# EXECUTE FUNCTION log_product_delete();
CREATE TRIGGER
company=#
```

Рисунок 13 – Создание триггера

1.4.3 Триггер проверки размера скидки

Не допустить скидку больше 50%

```
company=# CREATE OR REPLACE FUNCTION check_min_price()
company-# RETURNS TRIGGER AS $$
company$# BEGIN
company$#   IF NEW.unit_price < NEW.delivery_price * 0.5 THEN
company$#     RAISE EXCEPTION 'Ошибка: скидка превышает 50%% от цены доставки.';
company$#   END IF;
company$#   RETURN NEW;
company$# END;
company$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
company=#
```

Рисунок 14 – Создание функции триггера

```
company=# CREATE TRIGGER trg_check_min_price
company-# BEFORE INSERT OR UPDATE ON supply_details
company-# FOR EACH ROW
company-# EXECUTE FUNCTION check_min_price();
CREATE TRIGGER
company=#
```

Рисунок 15 – Создание триггера

```
company=# INSERT INTO supply_details (
company(#   product_id, supply_id, delivery_price, quantity, remaining_quantity,
company(#   unit_measure, weight_per_unit, expiration_date, unit_price, comments
company(# )
company-# VALUES (
company(#   1, 1001, 100, 10, 10,
company(#   'кг', 1, '2024-12-31', 40, 'Слишком дёшево!'  -- unit_price < 50
company(# );
ERROR:  Ошибка: скидка превышает 50% от цены доставки.
```

Рисунок 17 – Тестирование ввода некорректных данных

```
company=# INSERT INTO supply_details (
company(#   product_id, supply_id, delivery_price, quantity, remaining_quantity,
company(#   unit_measure, weight_per_unit, expiration_date, unit_price, comments
company(# )
company-# VALUES (
company(#   1, 1001, 100, 10, 10,
company(#   'кг', 1, '2024-12-31', 60, 'Нормальная скидка'
company(# );
INSERT 0 1
company=#
```

Рисунок 18 – Тестирование ввода корректных данных

1.4.4 Триггер автоматического уменьшения remaining_quantity после оформления заказа

Автоматическое уменьшение remaining_quantity после оформления заказа
Когда создаётся строка в order_details (т.е. оформлен заказ на товар), мы автоматически уменьшаем remaining_quantity в supply_details — на то количество, которое заказали.

функция триггера
CREATE OR REPLACE FUNCTION decrease_remaining_quantity()
RETURNS TRIGGER AS $$
BEGIN
  UPDATE supply_details
  SET remaining_quantity = remaining_quantity - NEW.quantity

```
  WHERE supply_details_id = NEW.supply_details_id;

  RETURN NEW;
END;
```

$$ LANGUAGE plpgsql;

создание триггера

```
company=# CREATE TRIGGER trg_decrease_remaining_quantity
company-# AFTER INSERT ON order_details
company-# FOR EACH ROW
company-# EXECUTE FUNCTION decrease_remaining_quantity();
CREATE TRIGGER
company=#
```

Рисунок 19 – Создание триггера

```
company=# SELECT remaining_quantity FROM supply_details WHERE supply_details_id = 1;
 remaining_quantity
--------------------
                 56
(1 row)

company=# INSERT INTO order_details (
company(#   supply_details_id, order_id, total_price, quantity, unit_price, unit_measure
company(# )
company-# VALUES (
company(#   1, 1, 1000, 5, 200, 'кг'
company(# );
INSERT 0 1
company=# SELECT remaining_quantity FROM supply_details WHERE supply_details_id = 1;
 remaining_quantity
--------------------
                 51
(1 row)

company=#
```

Рисунок 20 – Проверка работы триггера

1.4.5 Триггер контроля остатка перед заказом

Перед вставкой строки в order_details проверяем, что количество товара (quantity) не превышает текущий остаток (remaining_quantity) в соответствующей партии (supply_details).

CREATE OR REPLACE FUNCTION check_stock_before_order()

```
RETURNS TRIGGER AS $$
DECLARE
  current_stock INTEGER;
BEGIN
  -- Получаем текущий остаток товара
  SELECT remaining_quantity
  INTO current_stock
  FROM supply_details
  WHERE supply_details_id = NEW.supply_details_id;

  -- Проверка
  IF NEW.quantity > current_stock THEN
    RAISE EXCEPTION 'Ошибка: запрашиваемое количество превышает остаток на складе (% > %)', NEW.quantity, current_stock;
  END IF;

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
company=# CREATE TRIGGER trg_check_stock_before_order
company-# BEFORE INSERT ON order_details
company-# FOR EACH ROW
company-# EXECUTE FUNCTION check_stock_before_order();
CREATE TRIGGER
company=#
```

Рисунок – Создание триггера

```
company=# INSERT INTO order_details (
company(#   supply_details_id, order_id, total_price, quantity, unit_price, unit_measure
company(# )
company-# VALUES (
company(#   1, 1, 20000, 9999, 200, 'кг' -- Преднамеренно превышаем остаток
company(# );
ERROR:  Ошибка: запрашиваемое количество превышает остаток на складе (9999 > 51)
CONTEXT:  PL/pgSQL function check_stock_before_order() line 13 at RAISE
company=#
```

Рисунок – Тестирование триггера

1.4.6 Триггер проверка даты доставки

При вставке или обновлении строки в orders, мы не разрешаем, чтобы delivery_date была раньше order_date.

```
CREATE OR REPLACE FUNCTION validate_delivery_date()
        RETURNS TRIGGER AS $$
        BEGIN
          IF NEW.delivery_date IS NOT NULL AND NEW.delivery_date <
NEW.order_date THEN
            RAISE EXCEPTION 'Ошибка: дата доставки (% ) не может быть
раньше даты заказа (%).',
              NEW.delivery_date, NEW.order_date;
          END IF;
          RETURN NEW;
        END;
                                    $$ LANGUAGE plpgsql;
```

```
company=# CREATE TRIGGER trg_validate_delivery_date
company-# BEFORE INSERT OR UPDATE ON orders
company-# FOR EACH ROW
company-# EXECUTE FUNCTION validate_delivery_date();
CREATE TRIGGER
company=#
```

Рисунок – Создание триггера

```
company=# INSERT INTO orders (
company(#   customer_id, order_date, delivery_date,
company(#   payment_status, processing_status, manager_id, branch_id
company(# )
company-# VALUES (
company(#   1, '2024-04-27', '2024-04-25', -- \U+274C доставка раньше заказа
company(#   'ожидание', 'новый', 1, 1
company(# );
ERROR:  Ошибка: дата доставки (2024-04-25 ) не может быть раньше даты заказа (2024-04-27).
CONTEXT:  PL/pgSQL function validate_delivery_date() line 4 at RAISE
company=#
```

Рисунок – тестирование триггера

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы были успешно освоены практические навыки работы с хранимыми процедурами, функциями и триггерами в СУБД PostgreSQL. Разработанные три процедуры продемонстрировали возможность инкапсуляции сложной бизнес-логики на уровне базы данных, что позволяет повысить производительность приложений за счет уменьшения количества обращений к БД и переноса вычислений на сервер.

Созданные семь оригинальных триггеров реализовали важные аспекты контроля целостности данных и автоматизации бизнес-процессов гостиницы. Триггеры обеспечивают проверку возрастных ограничений для гостей, контроль дат бронирования и уборки, автоматическое применение акций, обновление статусов номеров и создание задач для персонала. Особое внимание было уделено обработке ошибок и созданию информативных сообщений для пользователей.

Практическая работа подтвердила преимущества использования триггеров для поддержания согласованности данных и реализации сложных бизнес-правил на уровне базы данных. Особенно ценным оказалось применение триггеров типа BEFORE для валидации данных и AFTER для реализации каскадных изменений. Также была продемонстрирована эффективность комбинирования процедур и триггеров для создания комплексных решений.

Выполнение дополнительных заданий позволило углубить понимание особенностей работы с переменными в PL/pgSQL, обработкой исключений и транзакциями. Разработанные объекты базы данных полностью соответствуют требованиям варианта задания и готовы к использованию в реальной системе управления гостиницей.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация PostgreSQL [Электронный ресурс] // Официальный сайт PostgreSQL. 1996-2025. URL: https://www.postgresql.org/docs/13/index.html (дата обращения: 20.03.2025).

2. Документация pgAdmin 4 PostgreSQL [Электронный ресурс] // Официальный сайт pgAdmin. URL: https://www.pgadmin.org/docs/pgadmin4/latest/ (дата обращения: 21.03.2025)