Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение
высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчет

по лабораторной работе № 3
“Процедуры, функции, триггеры в PostgreSQL”
по дисциплине **“Проектирование и реализация баз данных”**

Автор: Бускина Алия

Факультет: ИКТ

Группа: К32421

Преподаватель: Говорова М.М.

Санкт-Петербург, 2023

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

Оборудование: компьютерный класс.

Программное обеспечение: СУБД PostgreSQL, SQL Shell (psql).

Практическое задание:

Варинат 9. 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.

Задание 4. Создайте хранимую процедуру:

- для снижения цены на заданный процент для товаров, у которых срок пребывания на складе превысил заданный норматив.
- для расчета стоимости всех партий товаров, проданных за прошедшие сутки.

Задание 5. Создать необходимые триггеры.

Функции:

1) get_buyer_last_order (buyer_id) – функция, которая возвращает дату и сумму последнего заказа покупателя, а также название его компании по “buyer_id”

Текст функции:

```
CREATE OR REPLACE FUNCTION get_buyer_last_order(buyer_id varchar)
RETURNS TABLE(
 buyer_name varchar,
 last_order_date date,
 total_order bigint
)
AS $$
BEGIN
 RETURN QUERY
  SELECT
   buyer.buyer_name,
   MAX(orders.order_date) AS last_order_date,
   CAST(SUM(order_composition.amount_of_product * order_composition.cost_for_sale) AS
bigint) AS total_order
  FROM
   buyer
   JOIN orders ON buyer.buyer_id = orders.buyer_id
   JOIN order_composition ON orders.order_id = order_composition.order_id
  WHERE
   buyer.buyer_id = get_buyer_last_order.buyer_id
  GROUP BY
   buyer.buyer_name;
END;
$$ LANGUAGE plpgsql;
```

Скрин создания функции:

```
wholesale_base=# CREATE OR REPLACE FUNCTION get_buyer_last_order(buyer_id varchar)
wholesale_base-# RETURNS TABLE(
wholesale_base(#   buyer_name varchar,
wholesale_base(#   last_order_date date,
wholesale_base(#   total_order bigint
wholesale_base(# )
wholesale_base-# AS $$
wholesale_base$# BEGIN
wholesale_base$#   RETURN QUERY
wholesale_base$#     SELECT
wholesale_base$#       buyer.buyer_name,
wholesale_base$#       MAX(orders.order_date) AS last_order_date,
wholesale_base$#       CAST(SUM(order_composition.amount_of_product * order_composition.cost_for_sale) AS bigint) AS total_order
wholesale_base$#     FROM
wholesale_base$#       buyer
wholesale_base$#       JOIN orders ON buyer.buyer_id = orders.buyer_id
wholesale_base$#       JOIN order_composition ON orders.order_id = order_composition.order_id
wholesale_base$#     WHERE
wholesale_base$#       buyer.buyer_id = get_buyer_last_order.buyer_id
wholesale_base$#     GROUP BY
wholesale_base$#       buyer.buyer_name;
wholesale_base$# END;
wholesale_base$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
```

Скрин работы функции:

```
wholesale_base=# select * from get_buyer_last_order('buyer10');
 buyer_name | last_order_date | total_order
------------+-----------------+-------------
 company_10 | 2022-03-22      |       40401
(1 row)
```

2) get_product_availability (product_id) – функция возвращает доступное количество товара по его product_id, а также его название и id поставщика (customer_id)

Текст функции:

```
CREATE OR REPLACE FUNCTION get_product_availability(product_id
character varying)
RETURNS TABLE (
  product_name character varying(100),
  customer_id character varying(100),
  available_stock bigint
) AS $$
BEGIN
  RETURN QUERY
  SELECT
    product.product_name,
    product.customer_id,
    SUM(purchase_composition.stock) AS available_stock
  FROM
    product
    JOIN purchase_composition ON product.product_id =
purchase_composition.product_id
  WHERE
    product.product_id = get_product_availability.product_id
  GROUP BY
    product.product_name,
    product.customer_id;
END; $$
LANGUAGE plpgsql;
```

Скрин создания функции:

```
wholesale_base=# CREATE OR REPLACE FUNCTION get_product_availability(product_id character varying)
wholesale_base-# RETURNS TABLE (
wholesale_base(#     product_name character varying(100),
wholesale_base(#     customer_id character varying(100),
wholesale_base(#     available_stock bigint
wholesale_base(# ) AS $$
wholesale_base$# BEGIN
wholesale_base$#     RETURN QUERY
wholesale_base$#     SELECT
wholesale_base$#         product.product_name,
wholesale_base$#         product.customer_id,
wholesale_base$#         SUM(purchase_composition.stock) AS available_stock
wholesale_base$#     FROM
wholesale_base$#         product
wholesale_base$#         JOIN purchase_composition ON product.product_id = purchase_composition.product_id
wholesale_base$#     WHERE
wholesale_base$#         product.product_id = get_product_availability.product_id
wholesale_base$#     GROUP BY
wholesale_base$#         product.product_name,
wholesale_base$#         product.customer_id;
wholesale_base$# END; $$
wholesale_base-# LANGUAGE plpgsql;
CREATE FUNCTION
```

Скрин работы функции:

```
wholesale_base=# select * from get_product_availability('product36');
 product_name | customer_id | available_stock
--------------+-------------+-----------------
 orange       | customer6   |              15
(1 row)
```

Процедуры:

1) reduce_price (pct, date)– процедура, которая снижает цену на определенный процент для товаров, у которых срок хранения на базе превышает норматив

   Сначала создадим таблицу, где будут отображаться изменения на цену товаров:

```
CREATE TABLE product_price_history (
   id SERIAL PRIMARY KEY,
   product_id varchar REFERENCES product(product_id),
   old_price NUMERIC(10,2),
   new_price NUMERIC(10,2),
   date_changed TIMESTAMP DEFAULT NOW()
);
```

```
CREATE TABLE product_price_history (
    id SERIAL PRIMARY KEY,
    product_id varchar REFERENCES product(product_id),
    old_price NUMERIC(10,2),
    new_price NUMERIC(10,2),
    date_changed TIMESTAMP DEFAULT NOW()
);
```

   Далее создаем триггерную функцию и триггер, которые будут автоматически заполнять эту таблицу:

```
CREATE OR REPLACE FUNCTION add_to_price_history() RETURNS
TRIGGER AS $$
BEGIN
   INSERT INTO price_history (product_id, old_cost_ed, new_cost_ed,
changed_at)
   VALUES (OLD.product_id, OLD.cost_ed, NEW.cost_ed, NOW());
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER price_change_trigger
AFTER UPDATE ON product
FOR EACH ROW
WHEN (OLD.cost_ed != NEW.cost_ed)
EXECUTE FUNCTION add_to_price_history();
```

```
CREATE OR REPLACE FUNCTION add_to_price_history() RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO price_history (product_id, old_cost_ed, new_cost_ed, changec
    VALUES (OLD.product_id, OLD.cost_ed, NEW.cost_ed, NOW());
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER price_change_trigger
AFTER UPDATE ON product
FOR EACH ROW
WHEN (OLD.cost_ed != NEW.cost_ed)
EXECUTE FUNCTION add_to_price_history();
```

Создаем процедуру:

```
CREATE OR REPLACE PROCEDURE reduce_price1(pct NUMERIC, days INTEGER)
AS $$
BEGIN
  IF pct > 1 THEN
    pct = pct/100;
  END IF;

  UPDATE product
  SET cost_ed = CASE WHEN cost_ed * (1 - pct) < 10 THEN 0 ELSE cost_ed * (1 -
pct) END
  FROM purchase_composition
  JOIN purchase ON purchase_composition.purchase_id =
purchase.purchase_id
  WHERE
    purchase_composition.deadline_to_keep - purchase.shipment_date  <=
days
    AND product.product_id = purchase_composition.product_id
    AND cost_ed * (1 - pct) > 0;
END;
$$ LANGUAGE plpgsql;
```

```
wholesale_base=# CREATE OR REPLACE PROCEDURE reduce_price1(pct NUMERIC, days INTEGER) AS $$
wholesale_base$# BEGIN
wholesale_base$#     IF pct > 1 THEN
wholesale_base$#         pct = pct/100;
wholesale_base$#     END IF;
wholesale_base$#
wholesale_base$#     UPDATE product
wholesale_base$#     SET cost_ed = CASE WHEN cost_ed * (1 - pct) < 10 THEN 0 ELSE cost_ed * (1 - pct) END
wholesale_base$#     FROM purchase_composition
wholesale_base$#     JOIN purchase ON purchase_composition.purchase_id = purchase.purchase_id
wholesale_base$#     WHERE
wholesale_base$#         purchase_composition.deadline_to_keep - purchase.shipment_date  <= days
wholesale_base$#         AND product.product_id = purchase_composition.product_id
wholesale_base$#         AND cost_ed * (1 - pct) > 0;
wholesale_base$# END;
wholesale_base$# $$ LANGUAGE plpgsql;
CREATE PROCEDURE
```

Вызываем процедуру с параметрами 10% и 7 дней:

```
CALL reduce_price (10,7)
```

```
wholesale_base=# CALL reduce_price(10, 7);
CALL
```

Вызываем таблицу, где будут отображены все изменения цен на товары:

```
wholesale_base=# call reduce_price1(10,7);
CALL
wholesale_base=# select * from price_history;
 id | old_cost_ed | new_cost_ed |         changed_at         | product_id
----+-------------+-------------+----------------------------+------------
  1 |     1190.00 |     1071.00 | 2023-04-17 01:17:48.913783 | product89
  2 |      100.00 |       90.00 | 2023-04-17 01:17:48.913783 | product01
  3 |      100.00 |       90.00 | 2023-04-17 01:17:48.913783 | product02
  4 |      171.00 |      154.00 | 2023-04-17 01:17:48.913783 | product67
  5 |      100.00 |       90.00 | 2023-04-17 01:17:48.913783 | product68
  6 |     1190.00 |     1071.00 | 2023-04-17 01:17:48.913783 | product99
  7 |     1190.00 |     1071.00 | 2023-04-17 01:17:48.913783 | product100
  8 |      360.00 |      324.00 | 2023-04-17 01:17:48.913783 | product109
  9 |      360.00 |      324.00 | 2023-04-17 01:17:48.913783 | product110
 10 |      110.00 |       99.00 | 2023-04-17 01:17:48.913783 | product10
 11 |      115.00 |      104.00 | 2023-04-17 01:17:48.913783 | product1
 12 |      250.00 |      225.00 | 2023-04-17 01:17:48.913783 | product12
 13 |      175.00 |      158.00 | 2023-04-17 01:17:48.913783 | product13
 14 |      150.00 |      135.00 | 2023-04-17 01:17:48.913783 | product14
 15 |      171.00 |      154.00 | 2023-04-17 01:17:48.913783 | product16
 16 |      168.00 |      151.00 | 2023-04-17 01:17:48.913783 | product17
 17 |      131.00 |      118.00 | 2023-04-17 01:17:48.913783 | product24
 18 |      293.00 |      264.00 | 2023-04-17 01:17:48.913783 | product25
 19 |      181.00 |      163.00 | 2023-04-17 01:17:48.913783 | product26
 20 |      279.00 |      251.00 | 2023-04-17 01:17:48.913783 | product28
 21 |      257.00 |      231.00 | 2023-04-17 01:17:48.913783 | product36
 22 |      188.00 |      169.00 | 2023-04-17 01:17:48.913783 | product38
 23 |      130.00 |      117.00 | 2023-04-17 01:17:48.913783 | product50
 24 |      247.00 |      222.00 | 2023-04-17 01:17:48.913783 | product51
 25 |      275.00 |      248.00 | 2023-04-17 01:17:48.913783 | product85
 26 |      275.00 |      248.00 | 2023-04-17 01:17:48.913783 | product88
 27 |      360.00 |      324.00 | 2023-04-17 01:17:48.913783 | product122
 28 |      324.00 |      292.00 | 2023-04-17 01:17:48.913783 | product119
 29 |     1071.00 |      964.00 | 2023-04-17 16:43:29.692932 | product89
 30 |       90.00 |       81.00 | 2023-04-17 16:43:29.692932 | product01
 31 |       90.00 |       81.00 | 2023-04-17 16:43:29.692932 | product02
 32 |      154.00 |      139.00 | 2023-04-17 16:43:29.692932 | product67
 33 |       90.00 |       81.00 | 2023-04-17 16:43:29.692932 | product68
 34 |     1071.00 |      964.00 | 2023-04-17 16:43:29.692932 | product99
 35 |     1071.00 |      964.00 | 2023-04-17 16:43:29.692932 | product100
 36 |      324.00 |      292.00 | 2023-04-17 16:43:29.692932 | product109
 37 |      324.00 |      292.00 | 2023-04-17 16:43:29.692932 | product110
 38 |       99.00 |       89.00 | 2023-04-17 16:43:29.692932 | product10
 39 |      104.00 |       94.00 | 2023-04-17 16:43:29.692932 | product1
 40 |      225.00 |      203.00 | 2023-04-17 16:43:29.692932 | product12
 41 |      158.00 |      142.00 | 2023-04-17 16:43:29.692932 | product13
```

2) calculate_daily_sales() – функция, которая возвращает сумму стоимости всех товаров, проданных за предыдущие сутки.

Текст функции:

```
CREATE OR REPLACE FUNCTION calculate_daily_sales()
RETURNS TABLE (product_name varchar, sum numeric) AS $$
BEGIN
  RETURN QUERY
  SELECT p1.product_name, SUM(oc.cost_for_sale * oc.amount_of_product)
  FROM order_composition oc
  JOIN purchase_composition p ON p.id = oc.purchase_composition_id
  JOIN product p1 ON p1.product_id = p.product_id
  JOIN order_invoice oi ON oi.order_id = oc.order_id
  WHERE oi.payment_date >= CURRENT_DATE() - INTERVAL '1 day'
  GROUP BY p1.product_name;
END;
$$ LANGUAGE plpgsql;
```

```
wholesale_base=# CREATE OR REPLACE FUNCTION calculate_daily_sales()
wholesale_base-# RETURNS TABLE (product_name varchar, sum numeric) AS $$
wholesale_base$# BEGIN
wholesale_base$#     RETURN QUERY
wholesale_base$#     SELECT p1.product_name, SUM(oc.cost_for_sale * oc.amount_of_product)
wholesale_base$#     FROM order_composition oc
wholesale_base$#     JOIN purchase_composition p ON p.id = oc.purchase_composition_id
wholesale_base$#     JOIN product p1 ON p1.product_id = p.product_id
wholesale_base$#     JOIN order_invoice oi ON oi.order_id = oc.order_id
wholesale_base$#     WHERE oi.payment_date >= CURRENT_DATE - INTERVAL '1 day'
wholesale_base$#     GROUP BY p1.product_name;
wholesale_base$# END;
wholesale_base$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
```

Скрин работы функции:

```
wholesale_base=# SELECT * FROM calculate_daily_sales();
  product_name |  sum
 --------------+-------
  juice        |  3000
  melon        |  5200
  orange       | 12850
  stawberry    |  8525
  water melon  |  6216
 (5 rows)
```

Триггеры:

```
CREATE FUNCTION public.update_stock() RETURNS trigger
    LANGUAGE plpgsql
    AS $$
BEGIN
  UPDATE purchase_composition
  SET stock = stock - NEW.amount_of_products
  WHERE id = NEW.id;
  RETURN NEW;
END;
$$;
```

```
CREATE TRIGGER trg_purchase_composition BEFORE INSERT ON public.purchase_composition FOR EACH ROW
EXECUTE FUNCTION public.update_stock();
```

Данный триггер заполняет атрибут “stock” при добавлении строк в таблицу purchase_composition значениями атрибута “amount_of_product”

	id [PK] integer	purchase_id character varying (10)	product_id character varying (10)	amount_of_products integer	cost_for_sale integer	stock integer	deadline_to_keep date
1	17	purchase2	product122	5	360	5	2023-04-06
2	18	purchase2	product14	7	150	7	2023-04-06
3	19	purchase2	product26	2	181	2	2023-04-06
4	20	purchase2	product28	5	279	5	2023-04-06
5	21	purchase2	product110	4	360	4	2023-04-06

Вывод:

Обе процедуры работают с данными в базе данных и изменяют их.

Процедура reduce_price1 принимает два аргумента: pct - процент скидки в виде числа, и days - количество дней, в течение которых нужно применить скидку. Если значение pct меньше или равно 0, то никаких изменений не происходит. Если после применения скидки значение cost_ed становится меньше 10, то оно приравнивается к 0.

Функция calculate_daily_sales возвращает таблицу, содержащую два столбца: product_name - наименование продукта, и sum - общая сумма продаж за предыдущие сутки. Для этого функция производит выборку данных из таблиц базы данных и суммирует стоимость продаж для каждого продукта, учитывая только те продажи, оплаченные за предыдущие сутки.