|  |  |  |
| --- | --- | --- |
| **№** | **ВОПРОС** | **ОТВЕТ** |
| **1** | Заполнить таблицы базы данных, созданные на предыдущей лабораторной работе. | INSERT INTO counterparties (counterparty\_name, tin, address, city) VALUES  ('ООО "Ромашка"', '123456789012345', 'ул. Цветочная, 1', 'Москва'),  ('ИП Петров', '987654321098765', 'пр. Ленина, 10', 'Санкт-Петербург'),  ('АО "Солнышко"', '555555555555555', 'ул. Солнечная, 5', 'Екатеринбург');  INSERT INTO phone\_numbers (phone\_number, counterparty\_code) VALUES  ('+7 (495) 123-45-67', 1),  ('+7 (812) 987-65-43', 2),  ('+7 (343) 555-55-55', 3);  INSERT INTO accounts (bank\_name, bank\_account, corresponding\_account, bic, counterparty\_code) VALUES  ('Сбербанк', '40702810100000000000', '30101810100000000000', '044525225', 1),  ('Альфа-Банк', '40817810700000000000', '30101810000000000222', '044525225', 2),  ('ВТБ', '40817810800000000000', '30101810000000000333', '044525225', 3);  INSERT INTO products\_group (group\_name) VALUES  ('Овощи'),  ('Фрукты'),  ('Молочные продукты');  INSERT INTO products (product\_name, article, manufacturer, group\_code) VALUES  ('Помидоры', 'ART123', 'ООО "Агрофирма"', 1),  ('Яблоки', 'ART456', 'ЗАО "Сады России"', 2),  ('Молоко', 'ART789', 'ПАО "Молочная ферма"', 3);  INSERT INTO transactions (transaction\_date, price, quantity, transaction\_type, product\_code, supplier\_code, customer\_code, account\_code) VALUES  ('2024-03-08 10:00:00+03', 100.00, 10.00, 'Покупка', 1, 1, 2, 1),  ('2024-03-08 11:00:00+03', 50.00, 20.00, 'Продажа', 2, 2, 1, 2),  ('2024-03-09 12:00:00+03', 200.00, 5.00, 'Покупка', 3, 3, 2, 3);  INSERT INTO payments (amount, payment\_date, transaction\_code, account\_code) VALUES  (1000.00, '2024-03-15 14:00:00+03', 1, 1),  (1000.00, '2024-03-16 15:00:00+03', 2, 2),  (1000.00, '2024-03-17 16:00:00+03', 3, 3); |
| **2** | Написать процедуру, возвращающую первые N номеров Товаров и их названий,  где N передается в качестве аргумента процедуры. | CREATE OR REPLACE PROCEDURE get\_first\_n\_products(  num\_products INT,  OUT product\_data refcursor  ) AS $$  BEGIN  OPEN product\_data FOR  SELECT product\_code, product\_name  FROM products  LIMIT num\_products;  END;  $$ LANGUAGE plpgsql; |
| **3** | Дополнить предыдущую процедуру дополнительной проверкой: при попытке вызова процедуры с неопределенным параметром или отрицательным параметром должно вызываться исключение с сообщением о недопустимости оставлять параметр неопределенным или указывать отрицательное значение. Ситуация с нулевым значением параметра должна обрабатываться корректно. | CREATE OR REPLACE PROCEDURE get\_first\_n\_products(  num\_products INT,  OUT product\_data refcursor  ) AS $$  BEGIN  -- Проверка на допустимость значения num\_products  IF num\_products IS NULL THEN  RAISE EXCEPTION 'Параметр num\_products не может быть неопределенным.';  ELSIF num\_products < 0 THEN  RAISE EXCEPTION 'Параметр num\_products не может быть отрицательным.';  END IF;  OPEN product\_data FOR  SELECT product\_code, product\_name  FROM products  LIMIT num\_products;  END;  $$ LANGUAGE plpgsql;  DO $$  DECLARE  product\_cursor refcursor;  product\_record record;  result\_code TEXT;  BEGIN  -- Тесты  BEGIN  CALL get\_first\_n\_products(3, product\_cursor);  LOOP  FETCH product\_cursor INTO product\_record;  EXIT WHEN NOT FOUND;  RAISE NOTICE 'Product Code: %, Product Name: %', product\_record.product\_code, product\_record.product\_name;  END LOOP;  CLOSE product\_cursor;  EXCEPTION WHEN OTHERS THEN  GET STACKED DIAGNOSTICS result\_code = RETURNED\_SQLSTATE;  RAISE NOTICE 'Ошибка при обработке запроса: % %', result\_code, SQLERRM;  END;  END $$; |
| **4** | **Вычислить среднюю цену в таблице Операции. Для каждой записи в таблице сделать следующую проверку:**   * + **если существующая цена больше средней, то устанавливаем цену, равную средней + задаваемый фиксированный процент;**   + **если существующая цена меньше средней, то устанавливаем цену, равную прежней + половина разницы между прежней и средней ценой:**   + **если существующая цена равна средней, устанавливаем цену равной средней цене.** | CREATE OR REPLACE PROCEDURE update\_transaction\_prices(  INOUT avg\_price DECIMAL,  IN fixed\_percentage DECIMAL DEFAULT 0.1  ) AS $$  DECLARE  transaction\_record RECORD;  transaction\_updated INTEGER;  BEGIN  -- Проверка на существование средней цены  IF avg\_price IS NULL THEN  RAISE EXCEPTION 'Средняя цена не определена.';  END IF;  -- Обновление цен  FOR transaction\_record IN SELECT transaction\_code, price FROM transactions LOOP  IF transaction\_record.price > avg\_price THEN  UPDATE transactions  SET price = avg\_price + avg\_price \* fixed\_percentage  WHERE transaction\_code = transaction\_record.transaction\_code;  ELSIF transaction\_record.price < avg\_price THEN  UPDATE transactions  SET price = transaction\_record.price + (avg\_price - transaction\_record.price) / 2  WHERE transaction\_code = transaction\_record.transaction\_code;  ELSE  UPDATE transactions  SET price = avg\_price  WHERE transaction\_code = transaction\_record.transaction\_code;  END IF;  -- Проверка на успешное выполнение UPDATE  GET DIAGNOSTICS transaction\_updated = ROW\_COUNT;  IF transaction\_updated = 0 THEN  RAISE EXCEPTION 'Обновление записи с transaction\_code = % не выполнено.', transaction\_record.transaction\_code;  END IF;  END LOOP;  END;  $$ LANGUAGE plpgsql; |
| **5** | **Адаптировать примеры из файла *Хранимые процедуры.pdf* для своей БД** | —  Нет смысла. Например, get\_first\_n\_products отлично сочетает в себе сразу две процедуры из этого файла: top\_10 и positive. Создание отдельной процедуры для проверки положительности нарушает KISS, а top\_10 не сочетается с DRY. |
| **7** | **Адаптировать примеры триггеров для своей БД.** | ALTER TABLE products ADD COLUMN last\_updated TIMESTAMP WITH TIME ZONE;  CREATE OR REPLACE FUNCTION update\_last\_updated\_products()  RETURNS TRIGGER AS $$  BEGIN  NEW.last\_updated = NOW();  RETURN NEW;  END;  $$ LANGUAGE plpgsql;  CREATE TRIGGER update\_last\_updated\_trigger  BEFORE UPDATE ON products  FOR EACH ROW  EXECUTE PROCEDURE update\_last\_updated\_products();  SELECT product\_name, last\_updated FROM products WHERE product\_code = 1;  UPDATE products SET product\_name = 'Абрикосы' WHERE product\_code = 1;  SELECT product\_name, last\_updated FROM products WHERE product\_code = 1; |
| **8** | **Для таблицы «heavenly\_body», написать триггеры проверки на появление значений образующих циклы. (mod035)** | CREATE OR REPLACE FUNCTION detect\_cycle()  RETURNS TRIGGER AS $$  DECLARE  current\_id INT;  path\_ids INT[];  BEGIN  IF TG\_OP = 'INSERT' OR (TG\_OP = 'UPDATE' AND NEW.satellite IS DISTINCT FROM OLD.satellite) THEN  current\_id := NEW.num\_heavenly\_body;  path\_ids := array[current\_id];  WITH RECURSIVE cycle\_path AS (  SELECT satellite, array\_append(array[num\_heavenly\_body], satellite) AS cycle\_path\_ids  FROM heavenly\_body  WHERE num\_heavenly\_body = current\_id  UNION ALL  SELECT hb.satellite, array\_append(cp.cycle\_path\_ids, hb.satellite)  FROM heavenly\_body hb  INNER JOIN cycle\_path cp ON hb.num\_heavenly\_body = cp.satellite  WHERE hb.satellite IS NOT NULL  )  SELECT INTO path\_ids cycle\_path\_ids  FROM cycle\_path  WHERE satellite = current\_id;  IF path\_ids IS NOT NULL THEN  RAISE EXCEPTION 'Цикл обнаружен: %', array\_to\_string(path\_ids, ', ');  END IF;  *--Проверка на цикл длиной 1, если satellite == num\_heavenly\_body*  IF NEW.satellite = NEW.num\_heavenly\_body THEN  RAISE EXCEPTION 'Цикл длиной 1 обнаружен для небесного тела %', NEW.num\_heavenly\_body;  END IF;  END IF;  RETURN NEW;  END;  $$ LANGUAGE plpgsql;  INSERT INTO heavenly\_body (num\_heavenly\_body, name\_heavenly\_body, satellite) VALUES (13, 'Тест', 3);  UPDATE heavenly\_body SET satellite = 2 WHERE num\_heavenly\_body = 1; |
| **9** | **Создать таблицу, в которой для каждого номера страны указывается текущее количество запусков КА данной страной, написать триггеры, заполняющие эту таблицу** | CREATE TABLE country\_launches (  num\_country INTEGER NOT NULL PRIMARY KEY REFERENCES country(num\_country),  launch\_count INTEGER DEFAULT 0  );  CREATE OR REPLACE FUNCTION update\_country\_launches\_delete()  RETURNS TRIGGER AS $$  DECLARE  country\_id INT;  BEGIN  SELECT num\_country INTO country\_id FROM space\_vehicle WHERE num\_space\_vehicle = OLD.num\_space\_vehicle;  IF country\_id IS NULL THEN  RETURN NULL; --Не найден номер страны  END IF;  -- Уменьшаем счётчик до удаления записи  UPDATE country\_launches SET launch\_count = launch\_count - 1 WHERE num\_country = country\_id;  IF (SELECT launch\_count FROM country\_launches WHERE num\_country = country\_id) = 0 THEN  DELETE FROM country\_launches WHERE num\_country = country\_id;  END IF;  RETURN OLD;  END;  $$ LANGUAGE plpgsql;  CREATE TRIGGER update\_country\_launches\_delete\_trigger  BEFORE DELETE ON space\_vehicle  FOR EACH ROW  EXECUTE PROCEDURE update\_country\_launches\_delete(); |