|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |

Институт Информационных технологий (ИТ)

Кафедра Инструментального и прикладного программного обеспечения

(ИиППО)

**ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ №11**

**по дисциплине**

«Технологии обработки транзакций клиент-серверных приложений»

Выполнил студент группы ИКБО-20-21 Фомичев Р.А.

Принял Маличенко С.В.

Москва 2024**ЗАДАНИЕ НА ПРАКТИЧЕСКУЮ РАБОТУ**

1. Функция empapi.run, написанная на языке Python, принимает параметр типа jsonb. Добавьте трансформацию, чтобы избежать преобразований в текстовый вид и обратно.
2. Отправляйте пользователю, совершившему покупку, письмо-подтверждение с указанием суммы. Для отправки пользуйтесь уже готовой функцией public.sendmail (посмотрите ее определение) или напишите свою. Посылать письмо внутри транзакции покупки неправильно: транзакция может быть оборвана по какой-либо причине, а письмо уже уйдет. Воспользуйтесь механизмом фоновых заданий: в транзакции добавляйте задание на отправку письма. В таком случае оно будет отправлено, только если транзакция завершится успешно.
3. По возможности, повторите задание на языке Java с использованием технологии ORM.

**РЕФЕРАТ**

Отчёт 13 страниц, 6 рисунков, 5 источников, 1 приложение.

POSTGRESQL, SQL, ТРАНЗАКЦИИ, СЕРВЕРНОЕ ПРОГРАММИРОВАНИЕ, ДОВЕРЕННЫЕ ЯЗЫКИ

Объектом разработки является база данных.

Цель работы – изучение и практическое применение концепции транзакций в контексте баз данных с использованием языков программирования.

В процессе работы производилось исследование концепции транзакций в контексте баз данных с использованием языков программирования.

Результатом являются сведения концепции транзакций в контексте баз данных с использованием языков программирования.

**ВВЕДЕНИЕ**

[**ВВЕДЕНИЕ** 6](#_Toc162600479)

[**1.** **ВЫПОЛНЕНИЕ ПРАКТИЧЕСКОЙ РАБОТЫ** 7](#_Toc162600480)

[**1.1 Выполнение задания 1** 7](#_Toc162600481)

[**1.2 Выполнение задания 2** 8](#_Toc162600482)

[**1.3 Выполнение задания 3** 10](#_Toc162600483)

[ЗАКЛЮЧЕНИЕ 11](#_Toc162600484)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 1](#_Toc162600485)

[ПРИЛОЖЕНИЕ А 2](#_Toc162600486)

**ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ**

БД – база данных;

СУБД – система управления базами данных.

**ВВЕДЕНИЕ**

В современном информационном обществе, где огромные объёмы данных являются ключевым ресурсом для бизнеса, науки и повседневной жизни, вопрос обеспечения безопасности и надёжности хранения данных становится более критическим, чем когда-либо. Одним из основных аспектов обеспечения безопасности и надёжности данных в СУБД является блокировки отношений и других объектов.

Целью данной практической работы является изучение и практическое применение методов оптимизации транзакционных запросов в базах данных.

1. **ВЫПОЛНЕНИЕ ПРАКТИЧЕСКОЙ РАБОТЫ**
   1. **Выполнение задания 1**

Была создана функция tmprun путем добавления в функцию empapi.run строки “TRANSFORM FOR TYPE jsonb”, позволяющая не конвертировать вручную text в jsonb. Обновленный код функции представлен в листинге 1 приложения А.

Была создана функция test\_function, которая выводит данные из запрашиваемой таблицы. Код данной функции представлен в лиситинге 2 приложения А.

Результат работы функции tmprun представлен на рисунке 1.

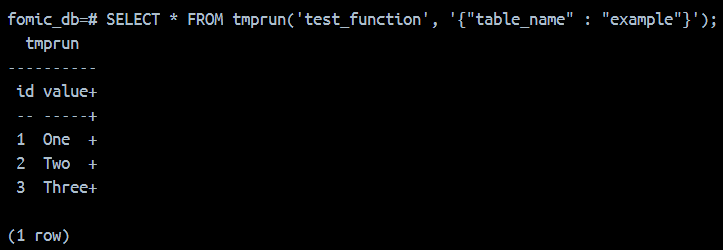


Рисунок 1 – Результат работы tmprun

* 1. **Выполнение задания 2**

Для выполнения данного задания использовалась реализация функции sendmail, которая позволяет пересылать сообщения с использованием smtp google.. Функция представлена в листинге 3 приложения А.

Был использован механизм фоновых заданий для реализации отложенной отправки сообщений. Задания записываются в отдельную таблицу и исполняются только по факту успешного завершения транзакции. В результате вызываемая функция внутри транзакции содержит только процесс записи задания в список.

Для демонстрации была создана функция checkout, выполняющаяся для проверки отправляемых заказов. Функция checkout представлена в листинге 4 приложения А.

Функция checkout вызывает внутри себя функцию before\_checkout с параметром user\_id(id пользователя, для которого производится проверка). Функция before\_checkout формирует задание по отправке сообщения с параметрами полученными из заказа. Функция before\_checkout представлена в листинге 5 приложения А.

Для помещения задания в очередь был произведен вызов функции checkout. Результат вызова представлен на рисунке 2.

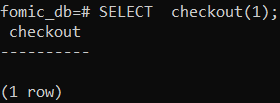


Рисунок 2 – Вызов функции checkout

Задание на отправку попало в очередь. Вывод списка очереди представлен на рисунке 3.

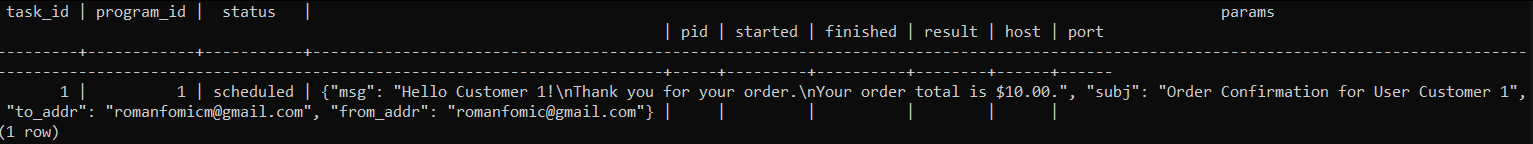


Рисунок 3 – Содержимое таблицы tasks

Функция execute\_tasks получает данные о текущих заданиях и выполняет их с помощью ранее описанной функции tmprun. В результате, было отправлено сообщение. Вызов функции execute\_task представлен на рисунке 4

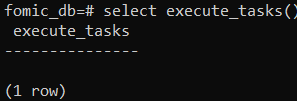


Рисунок 4 – Вызов функции execute\_tasks

Полученное сообщение представлено на рисунке 5.

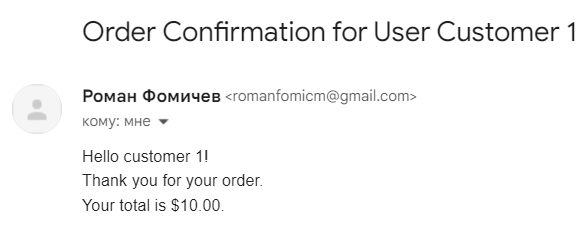


Рисунок 5 – Полученное сообщение

* 1. **Выполнение задания 3**

Для выполнения задания 3 было использовано java spring «Автосалон». Данное приложение использовало jpa репозитории взаимодействия с postgresql и модели для описания сущностей в приложении, примеры кода репозиториев и моделей представлены в листингах 6 и 7 соответственно.

Для рассылки сообщения использовался сервис DefaultEmailService. Код DefaultEmailService представлен в листинге 8.

Для вызова данной службы был изменен код контроллера отвечающего за обработку покупки автомобиля. Обновленный код представлен в листинге 9.

Для тестирования работоспособности был «куплен» автомобиль.

В результате было получено сообщение. Получение сообщения представлено на рисунке 6.

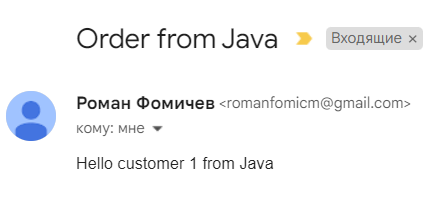


Рисунок 6 – Полученное сообщение

# ЗАКЛЮЧЕНИЕ

В ходе выполнения практической работы были получены практические навыки по применению различных языков серверного программирования для решения различных задач на стороне сервера. Также были изучены концепции транзакций в контексте баз данных с применением их на практике с использованием различных языков программирования.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. PostgreSQL: Официальная документация [Электронный ресурс] – URL: https://www.postgresql.org/docs/ (дата обращения: 14.02.2024).
2. PostgreSQL Tutorial for Beginners [Электронный ресурс] – URL: https://www.tutorialspoint.com/postgresql/index.htm (дата обращения: 14.02.2024).
3. PostgreSQL: Википедия [Электронный ресурс] – URL: https://en.wikipedia.org/wiki/PostgreSQL (дата обращения: 14.02.2024).
4. Лузанов П.В. и др. Postgres. Первое знакомство. [Электронный ресурс] – URL: https://postgrespro.ru/education/books/introbook (дата обращения: 22.02.2024)
5. Новиков Б. А. Лекции Основы технологий баз данных. [Электронный ресурс] – URL: https://postgrespro.ru/education/university/dbtech (дата обращения: 22.02.2024)

# ПРИЛОЖЕНИЕ А

**ФРАГМЕНТЫ КОДА РАЗРАБОТАННОГО ПРИЛОЖЕНИЯ**

Листинг 1 - Код функции tmprun

CREATE OR REPLACE FUNCTION tmprun(func text, params jsonb)

RETURNS text

TRANSFORM FOR TYPE jsonb

AS $python$

p = plpy.prepare("SELECT \* FROM " + plpy.quote\_ident(func) + "($1)", ["jsonb"])

r = p.execute([params])

cols = r.colnames()

collen = {col: len(col) for col in cols}

for i in range(len(r)):

for col in cols:

if len( str(r[i][col]) ) > collen[col]:

collen[col] = len( str(r[i][col]) )

res = ""

res += " ".join( [col.center(collen[col]," ") for col in cols]) + "\n"

res += " ".join( ["-"\*collen[col] for col in cols]) + "\n"

for i in range(len(r)):

res += " ".join( [str(r[i][col]).ljust(collen[col]," ") for col in cols]) + "\n"

return res

$python$ LANGUAGE plpython3u VOLATILE;

Листинг 2 - Код функции test\_function

CREATE OR REPLACE FUNCTION test\_function(params jsonb)

RETURNS TABLE(id INT, value TEXT)

AS $$

DECLARE

table\_name text;

query\_text text;

BEGIN

table\_name := params ->> 'table\_name';

query\_text := 'SELECT id, value::text FROM ' || quote\_ident(table\_name);

RETURN QUERY EXECUTE query\_text;

END;

$$ LANGUAGE plpgsql;

Листинг 3 - Код функции sendmail

CREATE OR REPLACE FUNCTION public.sendmail(from\_addr text, to\_addr text, subj text, msg text)

RETURNS void

LANGUAGE plpython3u

AS $function$

import smtplib

server = smtplib.SMTP('smtp.gmail.com', 587)

server.starttls()

server.login('romanfomic@gmail.com', 'uxnyugwremstfqfy')

server.sendmail(

from\_addr,

to\_addr,

"\r\n".join([

"From: %s" % from\_addr,

"To: %s" % to\_addr,

"Content-Type: text/plain; charset=\"UTF-8\"",

"Subject: %s" % subj,

"\r\n%s" % msg

]).encode('utf-8')

)

server.quit()

$function$;

Листинг 4 - Код функции checkout

CREATE FUNCTION public.checkout(user\_id bigint) returns void

AS $$

BEGIN

PERFORM before\_checkout(user\_id);

END;

$$ LANGUAGE plpgsql VOLATILE SECURITY DEFINER;

Листинг 5 - Код функции before\_checkout

CREATE OR REPLACE FUNCTION public.before\_checkout(user\_id bigint)

RETURNS void

AS $$

DECLARE

params jsonb;

BEGIN

SELECT jsonb\_build\_object(

'from\_addr', 'romanfomicm@gmail.com',

'to\_addr', c.email,

'subj', 'Order Confirmation for User ' || c.name,

'msg', format(

E'Hello %s!\nThank you for your order.\nYour order total is $%s.',

c.name,

total

)

)

INTO params

FROM customer c

JOIN (

SELECT so.customer\_id, SUM(i.total) AS total

FROM sales\_order so

JOIN item i ON so.order\_id = i.order\_id

WHERE so.customer\_id = user\_id

GROUP BY so.customer\_id

) AS order\_totals ON order\_totals.customer\_id = c.customer\_id;

PERFORM public.run\_program(

1,

params,

NULL,

NULL

);

END;

$$ LANGUAGE plpgsql VOLATILE;

Листинг 6 - Java код репозитория Rents

public interface SellRepo extends JpaRepository<Sell, Integer> {

Optional<List<Sell>> findBySnpassport(Long snpassport);

@Query(value = "SELECT \* FROM rents WHERE snpassport = ?1 AND end\_time IS NULL", nativeQuery = true)

Optional<Sell> findActiveRent(Long snpassport);

Optional<List<Sell>> findByVin(String vin);

}

Листинг 7 - Java код модели Sells

@AllArgsConstructor

@NoArgsConstructor

@Data

@Entity

@Table(name = "sells")

public class Sell {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "id\_sell")

private Integer idSell;

@JoinColumn(name = "snpassport")

private Long snpassport;

@JoinColumn(name = "vin")

private String vin;

@Column(name = "ending\_point")

private String endingPoint;

}

Листинг 8 - Java код класса DefaultEmailService

@Service

public class DefaultEmailService {

@Autowired

public JavaMailSender emailSender;

public void sendEmail(String toEmail, String subject, String body){

SimpleMailMessage message = new SimpleMailMessage();

message.setTo(toEmail);

message.setSubject(subject);

message.setText(body);

emailSender.send(message);

}

}

Листинг 9 - Java код обработчика, отвечающего за завершение аренды

@PostMapping("/buy/{id}")

public ResponseEntity<ExecutionResult<Sell>> startRent(@PathVariable Integer id, @RequestBody Sell sell){

ExecutionResult<Sell> result = sellService.buy(id, sell);

if (result.getErrorMessage() != null) {

return ResponseEntity.badRequest().body(result);

}

emailService.sendEmail("romanfomic@gmail.com", "Order from Java", "Hello customer 1 from Java");

return ResponseEntity.ok(result);

}

Листинг 10 – SQL код для выполнения задания 1

SELECT COUNT(\*) FROM sales\_order;

\timing on

DO $$

BEGIN

FOR i IN 1..1000 LOOP

EXECUTE 'SELECT avg(total) FROM SALES\_ORDER';

END LOOP;

END;

$$ LANGUAGE plpgsql;

DO $$

BEGIN

FOR i IN 1..1000 LOOP

PERFORM avg(total) FROM SALES\_ORDER;

END LOOP;

END;

$$ LANGUAGE plpgsql;

\timing off

Листинг 11 – SQL код для выполнения задания 2 практической работы

UPDATE SALES\_ORDER SET total = 11111 where order\_id = 70;

\timing on

DO $$

BEGIN

FOR i IN 1..100000 LOOP

EXECUTE 'SELECT \* FROM SALES\_ORDER WHERE total = 11111';

END LOOP;

END;

$$ LANGUAGE plpgsql;

DO $$

BEGIN

FOR i IN 1..100000 LOOP

PERFORM \* FROM SALES\_ORDER WHERE total = 11111;

END LOOP;

END;

$$ LANGUAGE plpgsql;

\timing off