

МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
Харківський національний університет радіоелектроніки

Кафедра системотехніки

ЗВІТ

з виконання завдань практичного заняття № 5
дисципліни «Проектування високонавантажених систем
зберігання даних»
на тему: «СТВОРЕННЯ ПРОЦЕДУР І ФУНКЦІЙ ДЛЯ
ВИСОКОНАВАНТАЖЕНИХ БАЗ ДАНИХ НА
ПЛАТФОРМІ СУБД MySQL»

Виконав
студент WILDAU - KHARKIV
Якунін Ігор

Перевірив
професор кафедри СТ
Колесник Л.В.

Харків, 2024

4.1 Мета заняття

- набуття практичних навичок зі створення збережених процедур (Stored Procedures) і функцій (Stored Functions) серверної частини високонавантаженої інформаційної системи;
- набуття практичних навичок з розробки SQL-запитів на вибірку й модифікацію даних, що використовуються в збережених процедурах і функціях, для забезпечення основних бізнес-процесів високонавантаженої інформаційної системи;
- набуття практичних навичок з розробки та використання курсорів (Cursors) у збережених процедурах;
- формування необхідних практичних умінь для аналізу плану виконання SQL-запитів за допомогою оператора EXPLAIN;
- формування необхідних практичних умінь для створення збережених процедур і функцій, з урахуванням особливостей роботи високонавантаженої інформаційної системи зберігання даних.

Завдання на самостійну роботу

Обрана така область – **«Сайт сервісного центру по ремонту техніки»**

Таблиця 5.1 – Обсяг виконаної роботи

№	Обсяг виконаних робіт	Кількість
1	Кількість створених збережених функцій, що використовують зв'язані базові таблиці БД (за допомогою тільки інструкції «WHERE», тільки інструкції «INNER JOIN», вкладеного запиту) і функції «CONCAT()», «GROUP_CONCAT()»;	17
2	Кількість створених збережених процедур, які використовують (викликають) збережені функції;	3
3	Кількість створених збережених процедур, що використовують зв'язані базові таблиці БД (за допомогою тільки інструкції «WHERE», тільки інструкції «INNER JOIN», вкладеного запиту) і функції «CONCAT()», «GROUP_CONCAT()»;	3
4	Кількість створених збережених процедур, що використовують оператори «IF...THEN...ELSEIF», «CASE v.1», « CASE v.2 »;	3
5	Кількість створених збережених процедур, які використовують (створюють й видаляють) тимчасову таблицю;	1
6	Кількість створених варіантів однієї (1) збереженої процедури, яка використовує тимчасову таблицю й курсор. Кожен варіант має реалізувати один з операторів циклу: 1) «WHILE», 2) «LOOP», 3) «REPEAT».	3

Таблиця 5.2 – Порівняльний аналіз можливостей процедур і функцій

№	Аналізовані параметри	Процедура	Функція
1	Синтаксис. Повернення результату.	Повертає результат через вихідні параметри (OUT) або змінює дані безпосередньо в базі.	Повертає одиночне значення через оператор RETURN.
2	Синтаксис. Формальні й фактичні параметри.	Підтримує параметри IN, OUT, INOUT.	Підтримує лише параметри IN.
3	Синтаксис. Виклик.	Викликається за допомогою CALL <procedure_name> (...).	Викликається як частина SQL-запиту, наприклад: SELECT <function_name> (...).
4	Результат, що повертається (результуюча множина, одиночне значення тощо).	Може повертати множину значень або результуючий набір через курсори, таблиці або змінні.	Повертає лише одиночне значення.
5	Можливі операції (операції над даними, створення набору даних тощо).	Може виконувати будь-які SQL-операції: SELECT, INSERT, UPDATE, DELETE	Може виконувати набір SQL-операцій (SELECT, також INSERT, UPDATE чи DELETE),
6	Підтримують наступні операції (зазначити: транзакції, SQL-інструкції SELECT, UPDATE, CURSOR тощо).	Підтримує транзакції, SQL-інструкції SELECT, UPDATE, INSERT, DELETE, курсори.	Підтримує транзакції, SQL-інструкції SELECT, UPDATE, INSERT, DELETE, курсори.
7	Взаємний виклик (процедура-функція, функція-процедура).	Може викликати інші процедури та функції.	Може викликати інші процедури та функції.
8	Статус програмного об'єкта (глобальний – для всіх БД, локальний – тільки для однієї БД тощо).	Локальний – доступний лише в рамках конкретної бази даних, в якій створено.	Локальний – доступний лише в рамках конкретної бази даних, в якій створено.
9	Порядок виконання сервером MySQL.	Виконуються окремо від запиту, що викликає процедуру (як окремий блок коду).	Виконуються як частина запиту (вбудовуються в SQL).
10	Фізичне зберігання (шлях, імена файлів та їх розширення).	Зберігаються в таблиці mysql.proc, як і функції, без окремого фізичного файлу.	Зберігаються в таблиці mysql.proc, як і процедури, без окремого фізичного файлу.

Таблиця 5.3 – Порівняльний аналіз можливостей процедур і тригерів

№	Аналізовані параметри	Процедура	Тригер
1	Синтаксис. Повернення результату.	Повертає результат через вихідні параметри (OUT) або змінює дані безпосередньо в базі.	Не повертає результат, виконується автоматично як частина дії (INSERT, UPDATE, DELETE).
2	Синтаксис. Формальні й фактичні параметри.	Підтримує параметри IN, OUT, INOUT.	Не має параметрів.
3	Синтаксис. Виклик.	Викликається явно за допомогою CALL <procedure_name> (...).	Викликається автоматично під час виконання певної дії (INSERT, UPDATE, DELETE) на таблиці, до якої він прив'язаний.
4	Результат, що повертається (результуюча множина, одиночне значення тощо).	Може повертати множину значень або результуючий набір через курсори, таблиці або змінні.	Не повертає жодних результатів.
5	Можливі операції (операції над даними, створення набору даних тощо).	Може виконувати будь-які SQL-операції: SELECT, INSERT, UPDATE, DELETE, виклик інших процедур та використання курсорів.	Може виконувати SQL-операції (INSERT, UPDATE, DELETE) для модифікації або логування дій.
6	Підтримують наступні операції (зазначити: транзакції, SQL-інструкції SELECT, UPDATE, CURSOR тощо).	Підтримує транзакції, SQL-інструкції SELECT, UPDATE, INSERT, DELETE, курсори.	Може включати SQL-інструкції (SELECT, INSERT, UPDATE, DELETE), але транзакції виконуються в контексті дії, що викликає тригер.
7	Взаємний виклик (процедура-тригер, тригер-процедура).	Може викликати тригер опосередковано через виконання дії, що викликає тригер.	Не може викликати процедури напряду.
8	Статус програмного об'єкта (глобальний – для всіх БД, локальний – тільки для однієї БД тощо).	Локальний – доступний лише в рамках конкретної бази даних, в якій створено.	Локальний – доступний лише для таблиці, до якої він прив'язаний.
9	Порядок виконання сервером MySQL.	Виконуються окремо, після явного виклику.	Виконуються автоматично перед або після (BEFORE, AFTER) вказаної дії (INSERT, UPDATE, DELETE).
10	Фізичне зберігання (шлях, імена файлів та їх розширення).	Зберігаються в таблиці mysql.proc.	Зберігаються в таблиці mysql.triggers.

Таблиця 5.4 - Переваги використання процедур (функцій)

№	Аналізовані параметри	Процедура	SQL-запит
1	Принцип виконання SQL-коду в СУБД MySQL	Виконуються на стороні сервера MySQL. Код зберігається і виконується безпосередньо сервером.	Виконуються як окремий запит, переданий серверу клієнтом.
2	Вплив на апаратні ресурси сервера	Використовують ресурси сервера під час виконання, однак можуть зменшити навантаження завдяки попередньо підготовленому коду.	Виконуються як окремий запит; ефективність залежить від обсягу даних та кількості запитів.
3	Вплив на мережний трафік	Зменшується мережевий трафік, оскільки передаються лише виклики процедур (функцій), а не великий обсяг SQL-коду.	Залежить від обсягу переданих запитів і даних; великі запити або множинні виклики можуть значно збільшувати трафік.
4	Можливість, щоб автоматизувати	Легко автоматизують складні послідовності дій (наприклад, транзакції, валідації, обробку даних).	Автоматизація можлива лише шляхом зовнішнього керування запитами (наприклад, через скрипти на стороні клієнта).
5	SQL-синтаксис (обмеження)	Можуть використовувати параметри, управляти курсорами, викликати інші процедури та виконувати складні обчислення.	Мають обмеження щодо багаторівневої логіки; складні обчислення потребують розбиття запитів.
6	Принцип зберігання SQL-коду	Код процедур (функцій) зберігається на сервері MySQL, у таблиці <code>mysql.proc</code> , що дозволяє централізоване управління та повторне використання.	Код зберігається на стороні клієнта, ускладнюючи його повторне використання та управління.

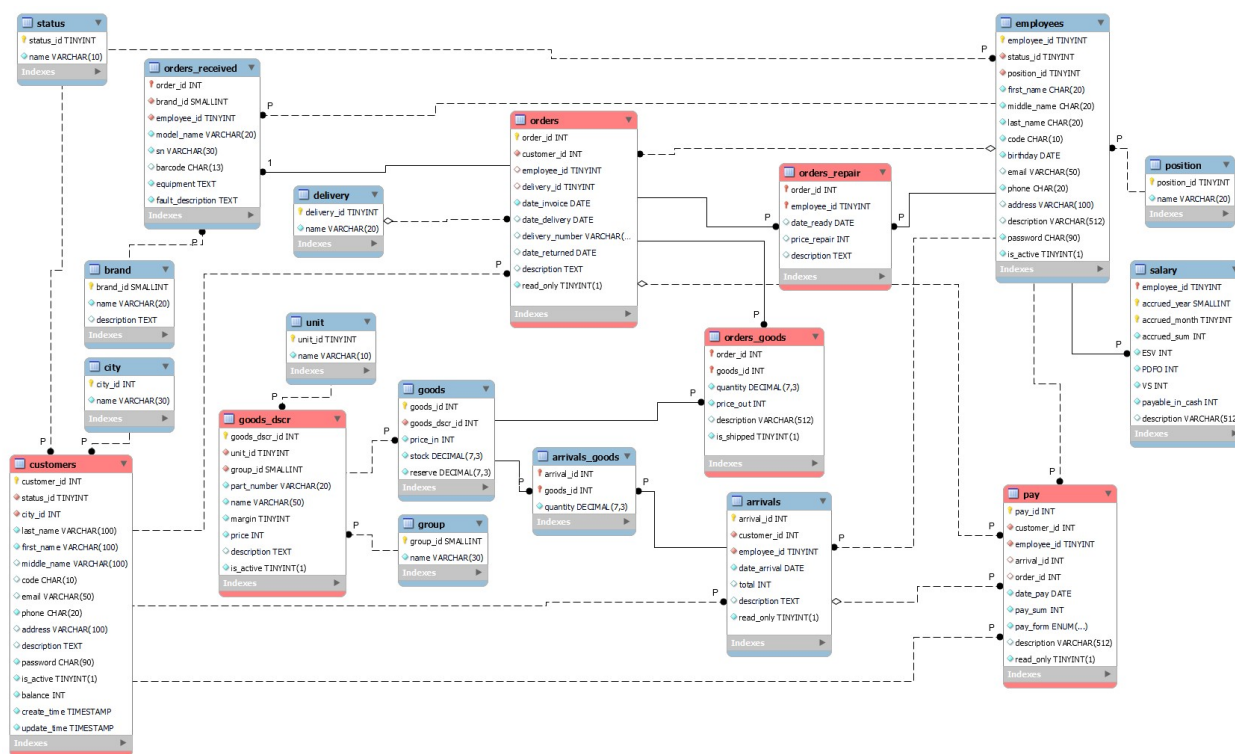


Рис. 5.1. Скріншот схеми фізичної моделі бази даних з таблицями типу InnoDB

Завдання 5.1. Для бази даних з таблицями InnoDB, створеної відповідно до завдання 1.3 практичного заняття № 1, розробити перелік збережених процедур, функцій(див. табл. 5.5), необхідних для забезпечення бізнес-функцій, на які впливає специфіка високонавантажених систем. Під час створення переліку врахувати результати виконання п.2.3.9 завдання 2.2.

Таблиця 5.5 – Процедури й функції високонавантаженої системи

№	Ім'я	Призначення процедури (функції)	Взаємозв'язок	Використовувані таблиці
1	get_customer_id	повертає customer_id по номеру телефону та хешу пароля		customers
2	get_cust_full_name	Повертає ПІБ користувача за його customer_id.		customers

3	get_cust_balance	Повертає Початковий БАЛАНС користувача		customers
4	get_customer_from_order	Повертає ім'я та прізвище користувача за id замовлення		customers orders
5	get_orders_from_customer	Повертає ПІБ користувача за customer_id та його замовлення	get_cust_full_name – повертає ПІБ	customers orders
6	get_order_goods_total	Повертає суму запчастин, які є в замовленні, за order_id		orders_goods
7	get_sum_repair	Повертає суму замовлення за ремонт за order_id		orders_repair
8	get_order_total_opt2	Повертає суму замовлення за id замовлення	get_order_goods_total , get_sum_repair	orders_goods, orders_repair
9	get_order_balance	Повертає Баланс замовлення за order_id	get_order_total_opt2	orders_goods, orders_repair, pay
10	get_sum_total_orders_for_customer_dop	Повертає суму замовлень користувача по customer_id	get_order_total_opt2	orders_goods, orders_repair, orders
11	get_total_sum_pay_for_customer	Повертає суму оплат користувача по customer_id		pay
12	get_total_balance_customer	Повертає Підсумковий Баланс користувача за customer_id	get_cust_balance, get_total_sum_pay_for_customer, get_sum_total_orders_for_customer_dop	orders_goods, orders_repair, orders, pay
13	get_order_date	Повертає дату		orders

	te	замовлення		
14 *	get_customer_card	Картка Клієнта	get_order_date, get_order_total_opt2, get_order_balance	orders_goods, orders_repair, orders, pay
15 *	get_act_customer_statement	Повертає відомість за активними клієнтами	get_sum_total_orders _for_customer_dop, get_sum_total_orders _for_customer_dop, get_total_balance_cus tomer	orders_goods, orders_repair, orders, pay, customers
16 *	get_customer_report	Відомість за активними користувачами	get_sum_total_orders _for_customer_dop, get_total_sum_pay_fo r_customer, get_total_balance_cus tomer	TEMPORARY TABLE + кypcop, orders_goods, orders_repair, orders, pay, customers
17	get_quantity_ available_ goods	Отримати кількість вільного товару		goods
18	get_quantity_ goods	повертає кількість запчастини в замовленні		orders_goods
19 *	set_orders_goods4	Списання запчастини в замовлення	get_quantity_available _goods, get_quantity_goods	TRANSACTION, Kypcop, orders, goods, orders_goods
20	func_order_ repair	закриває order_repair сьогоднішньою датою		orders, orders_repair, orders_received
20 *	set_orders_repair	закриває order_repair сьогоднішньою датою		orders, orders_repair, orders_received
21 *	set_date_orders	Ставимо дату замовлення - сьогодня		orders_repair, orders_goods Кypcopи
22 *	get_supplier_report	Відомість за активними постачальника ми		customers, pay, arrivals

23	get_supplier_card_start	повертає баланс постачальника на вказану дату		arrivals, pay, customers
24 *	get_supplier_card	Картка постачальника	get_supplier_card_start	arrivals, pay, customers

* - процедури

Завдання 5.2. Відповідно до переліку завдання 5.1 розробити збережені процедури й функції для бази даних з таблицями типу InnoDB і SQL-запити до них. Провести аналіз плану виконання SQL-запитів, що використовуються в збережених процедурах і функціях, за допомогою оператора EXPLAIN. Оцінити план виконання кожного SQL-запиту з висновком «неможливо оптимізувати» або «вимагає оптимізації». Підготувати стислі пропозиції з коректування коду SQL-запитів, схеми зв'язків, типів даних для зменшення часу запиту.

5.2.1. Отримати customer_id по номеру телефону та хешу пароля. Ноль, якщо не знайдено. Лістинг коду:

```
DELIMITER $$
SET GLOBAL log_bin_trust_function_creators = 1 $$
DROP FUNCTION IF EXISTS get_customer_id $$
CREATE FUNCTION get_customer_id (phone_number CHAR(20), password_hash CHAR(90))
RETURNS INT
COMMENT 'повертає customer_id по номеру телефону та хешу пароля. Ноль, якщо не знайдено.'
DETERMINISTIC
READS SQL DATA
BEGIN
    DECLARE id INT DEFAULT 0;

    SELECT customer_id INTO id FROM customers WHERE phone = phone_number and password = password_hash;

    RETURN id;
END $$
DELIMITER ;
```

```
5 • SELECT get_customer_id ('+380 08 739 83 69', '$pbkdf2-sha256$29000$bo2RUirFmPpeO0dojTFGqA$t9INJsMGIMICL7DgkJP4wWw5YVqodBIgTJVl82Xveyg');
```

get_customer_id ('+380 08 739 83 69', '\$pbkdf2-sha256\$29000\$bo2RUirFmPpeO0dojTFGqA\$t9INJsMGIMICL7DgkJP4wWw5YVqodBIgTJVl82Xveyg')
2

Рис. 5.2.1 Результат роботи функції get_customer_id

5.2.2. Отримати інформацію про користувача (ім'я та прізвище) за його customer_id. Лістинг коду:

```
DELIMITER $$
SET GLOBAL log_bin_trust_function_creators = 1 $$
DROP FUNCTION IF EXISTS get_cust_full_name $$
CREATE FUNCTION get_cust_full_name (id INT)
RETURNS VARCHAR(255)
COMMENT 'Повертає ПІБ користувача за його customer_id.'
DETERMINISTIC
READS SQL DATA

BEGIN
    DECLARE full_name VARCHAR(255) DEFAULT '';

    SELECT CONCAT(last_name, ' ', first_name, ' ', middle_name) INTO
    full_name FROM customers WHERE customer_id = id;

    RETURN full_name;
END $$
DELIMITER ;
```

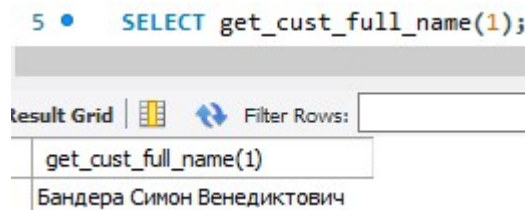


Рис. 5.2.2 Результат роботи функції get_cust_full_name

5.2.3. Отримати початковий баланс користувача за customer_id. Лістинг коду:

```
DELIMITER $$
SET GLOBAL log_bin_trust_function_creators = 1 $$
DROP FUNCTION IF EXISTS get_cust_balance $$
CREATE FUNCTION get_cust_balance (id INT)
RETURNS INT
COMMENT 'Повертає Початковий БАЛАНС користувача за його customer_id, або 0'
DETERMINISTIC
READS SQL DATA

BEGIN
    DECLARE cust_balance;

    SELECT balance INTO cust_balance FROM customers WHERE customer_id = id;

    RETURN cust_balance;
END $$
DELIMITER ;
```

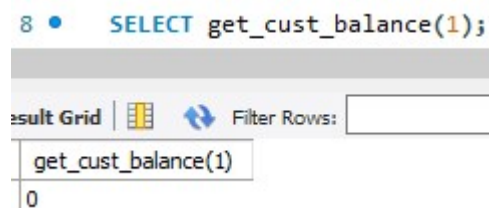


Рис. 5.2.3 Результат роботи функції get_cust_balance

5.2.4. Отримати ім'я та прізвище користувача за id замовлення. Лістинг коду:

```
DELIMITER $$
SET GLOBAL log_bin_trust_function_creators = 1 $$
DROP FUNCTION IF EXISTS get_customer_from_order $$
CREATE FUNCTION get_customer_from_order (id INT)
RETURNS VARCHAR(100)
COMMENT "Повертає ім'я та прізвище користувача за id замовлення"
DETERMINISTIC
READS SQL DATA

BEGIN
    DECLARE customer VARCHAR(100);

    SELECT CONCAT(first_name, ' ', last_name) INTO customer
    FROM orders o, customers c
    WHERE o.customer_id = c.customer_id and order_id = id;

    RETURN customer;
END $$
DELIMITER ;
```

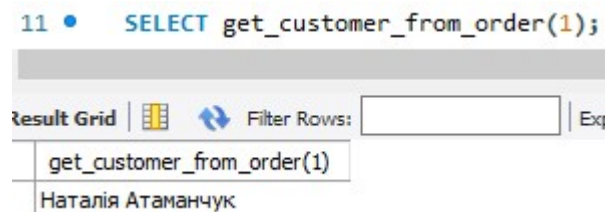


Рис. 5.2.4 Результат роботи функції get_customer_from_order

5.2.5. Отримати ПІБ користувача за customer_id та його замовлення.

Лістинг коду:

```
DELIMITER $$
SET GLOBAL log_bin_trust_function_creators = 1 $$
DROP FUNCTION IF EXISTS get_orders_from_customer $$
CREATE FUNCTION get_orders_from_customer (id INT)
RETURNS VARCHAR(255)
COMMENT "Повертає ПІБ користувача та його id замовлення по customer_id"
DETERMINISTIC
READS SQL DATA

BEGIN
    DECLARE orders VARCHAR(255);

    SELECT CONCAT(get_cust_full_name(id), ' - ', GROUP_CONCAT(order_id)) INTO
    orders
    FROM orders
    WHERE customer_id = id
    GROUP BY customer_id;

    RETURN orders;
END $$
DELIMITER ;
```

```
14 • SELECT get_orders_from_customer(1);
```

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
get_orders_from_customer(1)			
Бандера Симон Венедиктович - 1272,10004,10005,10006,10008,10009,10012,10019,10020,10030			

Рис. 5.2.5 Результат роботи функції get_customer_from_order

5.2.6. Підрахувати суму запчастин, які є в замовленні, за order_id.

Лістинг коду:


```
DELIMITER $$
SET GLOBAL log_bin_trust_function_creators = 1 $$
DROP FUNCTION IF EXISTS get_order_goods_total $$
CREATE FUNCTION get_order_goods_total (id INT)
RETURNS INT
COMMENT 'Повертає суму запчастин, які є в замовленні, по order_id'
DETERMINISTIC
READS SQL DATA
BEGIN
    DECLARE order_goods_total INT;


    SELECT ROUND(SUM(price_out * quantity)) INTO order_goods_total
    FROM orders_goods
    WHERE order_id = id;

    RETURN order_goods_total;
END $$
DELIMITER ;
```

```
37 • SELECT order_id, goods_id, quantity,
38         price_out, is_shipped
39 FROM orders_goods
40 WHERE order_id = 1;
```

Result Grid





Filter Rows:

Edit:

order_id	goods_id	quantity	price_out	is_shipped
1	1	1.000	2580	1
1	886	1.834	7230	1
1	5216	0.389	3831	1
1	8568	0.500	3300	1
NULL	NULL	NULL	NULL	NULL

```
17 • SELECT get_order_goods_total(1);
```

Result Grid	Filter Rows:
get_order_goods_total(1)	
18980	

Рис. 5.2.6 Результат роботи функції get_order_goods_total

5.2.7. Підрахувати суму замовлення за ремонт, за order_id. Лістинг

коду:

```
DELIMITER $$
SET GLOBAL log_bin_trust_function_creators = 1 $$
DROP FUNCTION IF EXISTS get_sum_repair $$
CREATE FUNCTION get_sum_repair (id INT)
RETURNS INT
COMMENT 'Повертає суму замовлення за ремонт за order_id, або 0'
DETERMINISTIC
READS SQL DATA

BEGIN
    DECLARE sum_repair INT DEFAULT 0;

    SELECT SUM(price_repair) INTO sum_repair FROM orders_repair
        WHERE order_id = id;

    RETURN sum_repair;
END $$
DELIMITER ;
```

```
11 • SELECT order_id, price_repair
12     FROM orders_repair
13     WHERE order_id = 1;
```

Result Grid	
order_id	price_repair
1	23000
1	37000

```
20 • SELECT get_sum_repair(1);
```

Result Grid	
get_sum_repair(1)	
60000	

Рис. 5.2.7 Результат роботи функції get_sum_repair

5.2.8. Підрахувати суму замовлення по id замовлення. Лістинг коду:

```
DELIMITER $$
SET GLOBAL log_bin_trust_function_creators = 1 $$
DROP FUNCTION IF EXISTS get_order_total_opt2 $$
CREATE FUNCTION get_order_total_opt2 (id INT)
RETURNS INT
COMMENT 'Повертає суму замовлення по id замовлення, оптимізовано!'
DETERMINISTIC
READS SQL DATA
BEGIN
    DECLARE order_total INT;

    SELECT ROUND(COALESCE(get_order_goods_total(id), 0) +
        COALESCE(get_sum_repair(id), 0)) INTO order_total;
    RETURN order_total;
END $$
DELIMITER ;
```

23 • `SELECT get_order_total_opt2(1);`

get_order_total_opt2(1)
78980

Рис. 5.2.8 Результат роботи функції `get_order_total_opt2`

5.2.9. Підрахувати баланс замовлення за `order_id` . Лістинг коду:

```
DELIMITER $$
SET GLOBAL log_bin_trust_function_creators = 1 $$
DROP FUNCTION IF EXISTS get_order_balance $$
CREATE FUNCTION get_order_balance (id INT)
RETURNS INT
COMMENT "Повертає Баланс замовлення за order_id"
DETERMINISTIC
READS SQL DATA

BEGIN
    DECLARE order_balance INT;

    SELECT SUM(pay_sum) - get_order_total_opt2(id)
    INTO order_balance
    FROM pay
    WHERE order_id = id;

    RETURN order_balance;
END $$
DELIMITER ;
```

28 • `SELECT order_id, SUM(pay_sum)`
 29 `FROM pay`
 30 `WHERE order_id = 1;`

order_id	SUM(pay_sum)
1	43929

26 • `SELECT get_order_balance(1);`

get_order_balance(1)
-35051

Рис. 5.2.9 Результат роботи функції `get_order_balance`

5.2.10. Отримати суму всіх замовлень користувача за `customer_id`.
 Лістинг коду:

```

DELIMITER $$
SET GLOBAL log_bin_trust_function_creators = 1 $$
DROP FUNCTION IF EXISTS get_sum_total_orders_for_customer_dop $$
CREATE FUNCTION get_sum_total_orders_for_customer_dop (id INT)
RETURNS INT
COMMENT "Повертає суму замовлень користувача по customer_id за допомоги
доп.функції"
DETERMINISTIC
READS SQL DATA

BEGIN
    DECLARE orders_total INT;

    SELECT SUM(get_order_total_opt2(order_id)) INTO orders_total
    FROM orders
    WHERE customer_id = id and date_returned IS NOT NULL;

    RETURN orders_total;
END $$
DELIMITER ;

```

```

21 • SELECT order_id, ROUND(quantity * price_out) as sum_spare, price_repair
22 FROM orders
23 LEFT JOIN orders_goods USING(order_id)
24 LEFT JOIN orders_repair USING(order_id)
25 WHERE customer_id = 1 and date_returned IS NOT NULL;

```

Result Grid Filter Rows: <input type="text"/> Export: Wrap Cell Content:			
order_id	sum_spare	price_repair	
1272	11221	104085	
1272	13521	104085	
10005	NULL	51000	
10005	NULL	37000	

```

29 • SELECT get_sum_total_orders_for_customer_dop(1);

```

Result Grid Filter Rows: <input type="text"/> Export: Wrap Cell	
get_sum_total_orders_for_customer_dop(1)	
216827	

Рис. 5.2.10 Результат роботи функції get_sum_total_orders_for_customer_dop

5.2.11. Отримати суму всіх оплат користувача за customer_id. Лістинг коду:

```

DELIMITER $$
SET GLOBAL log_bin_trust_function_creators = 1 $$
DROP FUNCTION IF EXISTS get_total_sum_pay_for_customer $$
CREATE FUNCTION get_total_sum_pay_for_customer (id INT)
RETURNS INT

```



```

COMMENT "Повертає суму оплат користувача по customer_id"
DETERMINISTIC
READS SQL DATA

BEGIN
    DECLARE pay_total INT;

    SELECT SUM(pay_sum) INTO pay_total
    FROM pay
    WHERE customer_id = id and order_id IS NOT NULL;

    RETURN pay_total;
END $$
DELIMITER ;

```

19 • SELECT pay_id, order_id, date_pay, pay_sum
 20 FROM pay
 21 WHERE customer_id = 1 and order_id IS NOT NULL

pay_id	order_id	date_pay	pay_sum
7325	1272	2024-06-18	69551
11001	1272	2024-09-20	50000
11002	10005	2024-11-06	83000
NULL	NULL	NULL	NULL

32 • SELECT get_total_sum_pay_for_customer(1);

get_total_sum_pay_for_customer(1)
202551

Рис. 5.2.11 Результат роботи функції get_total_sum_pay_for_customer

5.2.12. Отримати підсумковий Баланс користувача за customer_id.

Лістинг коду:

```

DELIMITER $$
SET GLOBAL log_bin_trust_function_creators = 1 $$
DROP FUNCTION IF EXISTS get_total_balance_customer $$
CREATE FUNCTION get_total_balance_customer (id INT)
RETURNS INT
COMMENT "Повертає Підсумковий Баланс користувача за customer_id"
DETERMINISTIC
READS SQL DATA

BEGIN
    DECLARE total_balance INT;

    SELECT get_cust_balance (id) +
           COALESCE(get_total_sum_pay_for_customer(id), 0) -
           COALESCE(get_sum_total_orders_for_customer_dop(id), 0)
    INTO total_balance;

```



```

RETURN total_balance;
END $$
DELIMITER ;

```

35 • SELECT get_total_balance_customer(1);

Result Grid | Filter Rows: | Exports:

get_total_balance_customer(1)
-14276

Рис. 5.2.12 Результат роботи функції get_total_balance_customer

5.2.13. Отримати дату відвантаження замовлення за order_id. Лістинг

коду:

```

DELIMITER $$
SET GLOBAL log_bin_trust_function_creators = 1 $$
DROP FUNCTION IF EXISTS get_order_date $$
CREATE FUNCTION get_order_date (id INT)
RETURNS DATE
COMMENT 'Повертає дату замовлення по id замовлення'
DETERMINISTIC
READS SQL DATA
BEGIN
    DECLARE order_date DATE;

    SELECT date_returned INTO order_date
    FROM orders
    WHERE order_id = id;

    RETURN order_date;
END $$
DELIMITER ;

```

38 • SELECT get_order_date(1);

Result Grid | Filter Rows:

get_order_date(1)
2024-09-07

Рис. 5.2.13 Результат роботи функції get_order_date

5.2.14. Отримати Картку клієнта. Лістинг коду:

```

DELIMITER $$
DROP PROCEDURE IF EXISTS get_customer_card $$
CREATE PROCEDURE get_customer_card (IN id INT)

COMMENT "Картка Клієнта - Повертає order_id, date_returned, order_total,
'Оплачено - pay_id, date_pay, date_pay;..', order_balance за customer_id"
DETERMINISTIC

```

READS SQL DATA

BEGIN

```
SELECT order_id, get_order_date(order_id) as 'Відвантажено', ,
       get_order_total_opt2(order_id) as order_total,
       GROUP_CONCAT(pay_id, ', ', date_pay, ', ', pay_sum SEPARATOR '; ')
as 'Оплачено - pay_id, date_pay, pay_sum;..',
       get_order_balance(order_id) as order_balance
FROM pay
WHERE customer_id = id and order_id IS NOT NULL
GROUP BY order_id;

END $$
DELIMITER ;
```

41 • CALL get_customer_card(1);

order_id	Відвантажено	order_total	Оплачено - pay_id, date_pay, pay_sum;..	order_balance
1272	2024-03-02	128827	7325, 2024-06-18, 69551; 11001, 2024-09-20, 50000	-9276
10005	2024-09-09	88000	11002, 2024-11-06, 83000	-5000

Рис. 5.2.14 Результат роботи процедури get_customer_card

5.2.15. Отримати відомість за активними клієнтами. Лістинг коду:

```
DELIMITER $$
DROP PROCEDURE IF EXISTS get_act_customer_statement $$
CREATE PROCEDURE get_act_customer_statement ()

COMMENT "Statement of customers - Повертає відомість за активними клієнтами"
DETERMINISTIC
READS SQL DATA

BEGIN

SELECT customer_id, balance,
       get_sum_total_orders_for_customer_dop(customer_id) as 'Відвантажено',
       get_total_sum_pay_for_customer(customer_id) as 'Оплачено',
       get_total_balance_customer(customer_id) as total_balance
FROM customers
WHERE get_sum_total_orders_for_customer_dop(customer_id) IS NOT NULL
      OR get_total_sum_pay_for_customer(customer_id) IS NOT NULL;

END $$
DELIMITER ;
```

44 • **CALL** get_act_customer_statement();

customer_id	balance	Відвантажено	Оплачено	total_balance
1	0	216827	202551	-14276
2	0	191543	67514	-124029
3	0	NULL	40131	40131
4	0	46364	21594	-24770
5	0	48066	108804	60738

14 11:56:35 CALL get_act_customer_statement() 8587 row(s) returned 1.609 sec / 10.907 sec

Рис. 5.2.15.1 Результат роботи процедури get_act_customer_statement

45 • **CALL** get_act_customer_statement2 ();

customer_id	start_balance	order_total	pay_total	total_balance
441	0	78980	43929	-35051
1735	0	NULL	65024	65024
4883	0	NULL	20162	20162
771	0	NULL	152902	152902
4526	0	NULL	102108	102108
1836	0	NULL	20310	20310
8922	0	NULL	73620	73620

113 22:06:09 CALL get_act_customer_statement2 () 6321 row(s) returned 0.109 sec / 0.000 sec

Рис. 5.2.15.2 Результат роботи процедури get_act_customer_statement2

Процедура get_act_customer_statement2 не використовує підрахунок проміжних сум з таблиць orders_goods та orders_repair. Вона бере дані з поля total таблиці orders. Швидкодія зростає в 100 разів! Лістинг коду процедури:

```
DELIMITER $$
DROP PROCEDURE IF EXISTS get_act_customer_statement2 $$
CREATE PROCEDURE get_act_customer_statement2 ()

COMMENT "Statement of customers - Повертає відомість за активними клієнтами"
DETERMINISTIC
READS SQL DATA

BEGIN
    SELECT o.customer_id,
           MAX(balance) as start_balance,
           SUM(total) as order_total,
           SUM(pay_sum) as pay_total,
           MAX(balance) - coalesce(SUM(total), 0) + coalesce(SUM(pay_sum), 0)
    as total_balance
    FROM orders o
    LEFT JOIN pay p ON o.order_id = p.order_id
    JOIN customers c ON c.customer_id = o.customer_id
```

```

GROUP BY o.customer_id;
END $$
DELIMITER ;

```

Подивимось, чи потрібна оптимізація процедури, використаємо EXPLAIN.

```

88 • SELECT o.customer_id,
89       MAX(balance) as start_balance,
90       SUM(total) as order_total,
91       SUM(pay_sum) as pay_total,
92       MAX(balance) - coalesce(SUM(total), 0) + coalesce(SUM(pay_sum), 0) as total_balance
93 FROM orders o
94 LEFT JOIN pay p ON o.order_id = p.order_id
95 JOIN customers c ON c.customer_id = o.customer_id
96 GROUP BY o.customer_id;

```

Tabular Explain

i	select_type	table	partiti...	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	o		ALL	fk_orders_customers_idx				9591	100.00	Using temporary
1	SIMPLE	c		eq_ref	PRIMARY			ss_innodb.o.customer...	1	100.00	
1	SIMPLE	p		ref	fk_pay_orders1_idx	fk_pay_orders1_idx	5	ss_innodb.o.order_id	1	100.00	

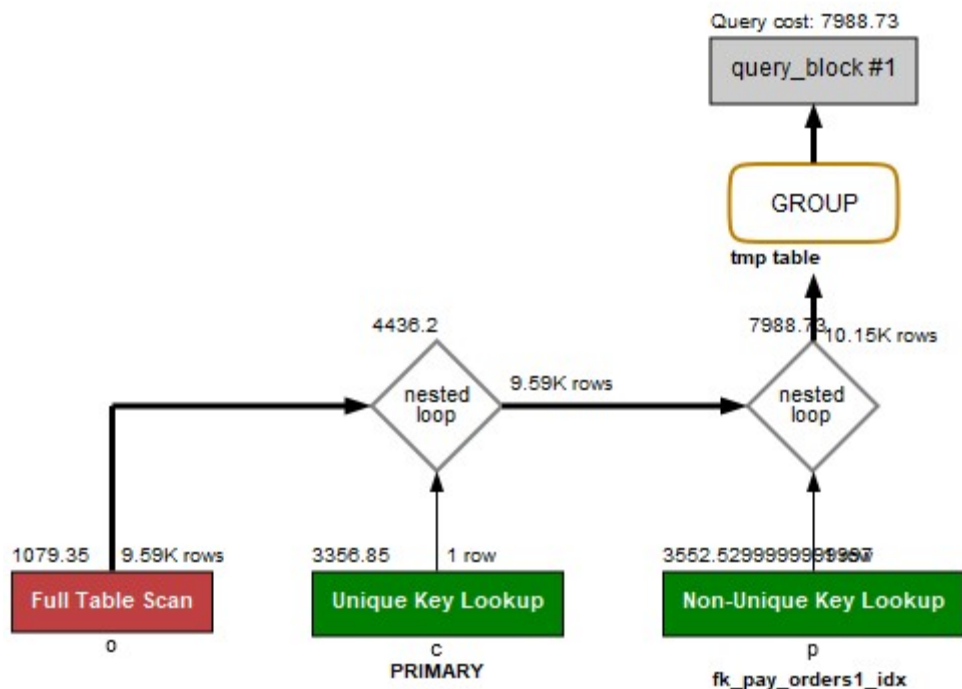


Рис. 5.2.15.3 Скриншот EXPLAIN запиту, що використовує процедура

Тут є Full Table Scan, але ж нам потрібно отримати інформацію зі всіх рядків таблиці і вивести її в табличному вигляді. Мені здається, що нині тут оптимізація не потрібна. Головне ми зробили – додали поле total до таблиці, це значно прискорило обробку даних таблиці.

5.2.16. Отримати відомість за активними клієнтами. Інший спосіб, з використанням **тимчасової таблиці та курсору**. Лістинг коду:

```

DELIMITER $$
DROP PROCEDURE IF EXISTS get_customer_report $$

```

```

CREATE PROCEDURE get_customer_report()

COMMENT 'Відомість за активними користувачами'
LANGUAGE SQL
DETERMINISTIC
MODIFIES SQL DATA

BEGIN

    DECLARE done INT DEFAULT 0;
    DECLARE var_customer_id INT;
    DECLARE var_balance INT;
    DECLARE var_total_shipped INT;
    DECLARE var_total_paid INT;
    DECLARE var_total_balance INT;

    -- Курсор для вибірки клієнтів
    DECLARE cur CURSOR FOR SELECT customer_id, balance FROM customers;

    -- Обробка ситуації, коли курсор повертає NULL
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    CREATE TEMPORARY TABLE IF NOT EXISTS temp_customer_report (
        customer_id INT,
        balance INT,
        total_shipped INT,
        total_paid INT,
        total_balance INT);

    OPEN cur;

    -- Обробляємо кожного клієнта по черзі
    read_loop: LOOP
        FETCH cur INTO var_customer_id, var_balance;
        IF done THEN
            LEAVE read_loop;
        END IF;

        SET var_total_shipped =
get_sum_total_orders_for_customer_dop(var_customer_id);
        SET var_total_paid = get_total_sum_pay_for_customer(var_customer_id);

        IF var_total_shipped IS NOT NULL OR var_total_paid IS NOT NULL THEN
            SET var_total_balance =
get_total_balance_customer(var_customer_id);
            INSERT INTO temp_customer_report (customer_id, balance,
total_shipped, total_paid, total_balance)
                VALUES (var_customer_id, var_balance, var_total_shipped,
var_total_paid, var_total_balance);
            END IF;
        END LOOP;

    CLOSE cur;
    SELECT * FROM temp_customer_report;
    DROP TEMPORARY TABLE IF EXISTS temp_customer_report;
END $$
DELIMITER ;

```

47 • **CALL** get_customer_report();

Result Grid | Filter Rows: | Export: | Wrap Cell

	customer_id	balance	total_shipped	total_paid	total_balance
1	1	0	216827	202551	-14276
2	2	0	191543	67514	-124029
3	3	0	NULL	40131	40131
4	4	0	46364	21594	-24770
5	5	0	48066	108804	60738
6	6	0	NULL	27704	27704

15 12:38:36 CALL get_customer_report() 8587 row(s) returned 9.703 sec / 0.000 sec

Рис. 5.2.16.1 Результат роботи процедури get_customer_report

48 • **CALL** get_customer_report_dec();

Result Grid | Filter Rows: | Export: | Wrap Cell

	customer_id	balance	total_shipped	total_paid	total_balance
1	1	0.00	216827.00	202551.00	-14276.00
2	2	0.00	191543.00	67514.00	-124029.00
3	3	0.00	NULL	40131.00	40131.00
4	4	0.00	46364.00	21594.00	-24770.00
5	5	0.00	48066.00	108804.00	60738.00
6	6	0.00	NULL	27704.00	27704.00

16 12:44:59 CALL get_customer_report_dec() 8587 row(s) returned 9.765 sec / 0.016 sec

Рис. 5.2.16.2 Результат роботи процедури get_customer_report_dec
(тип даних – decimal(10,2))

5.2.17. Отримати кількість вільного товару по goods_dscr_id (сумма кількох goods_id). Лістинг коду:



```
DELIMITER $$
DROP FUNCTION IF EXISTS get_quantity_available_goods $$
CREATE FUNCTION get_quantity_available_goods (id INT)
RETURNS INT
COMMENT "Отримати кількість вільного товару за goods_dscr_id "
DETERMINISTIC
READS SQL DATA

BEGIN
    DECLARE available_quantity INT;

    SELECT SUM(stock) - SUM(reserve) INTO available_quantity
    FROM goods
    WHERE goods_dscr_id = id;




    RETURN available_quantity;
END $$
DELIMITER ;
```


14 • SELECT goods_id, price_in,
15 stock, reserve
16 FROM goods
17 WHERE goods_dscr_id = 9999;

Result Grid |   Filter Rows:

	goods_id	price_in	stock	reserve
▶	9999	7212	4.000	4.000
	10002	7000	2.000	2.000
	10003	7500	3.000	2.000
*	NULL	NULL	NULL	NULL

51 • SELECT get_quantity_available_goods(9999);

result Grid |   Filter Rows: | Export: 

	get_quantity_available_goods(9999)
1	

Рис. 5.2.17 Результат роботи функції get_quantity_available_goods

5.2.18. Отримати кількість запчастини (за goods_id) в замовленні.

Лістинг коду:

```
DELIMITER $$
DROP FUNCTION IF EXISTS get_quantity_goods $$
CREATE FUNCTION get_quantity_goods (o_id INT, g_id INT)
RETURNS DECIMAL(7,3)
COMMENT "повертає кількість запчастини (за goods_id) в замовленні, або ноль,
якщо такої запчастини немає "
DETERMINISTIC
READS SQL DATA
BEGIN
    DECLARE quantity_goods DECIMAL(7,3);

    SELECT quantity INTO quantity_goods
    FROM orders_goods
    WHERE order_id = o_id AND goods_id = g_id;

    RETURN coalesce(quantity_goods, 0);
END $$
DELIMITER ;
```

37 • `SELECT order_id, goods_id, quantity,`
38 `price_out, is_shipped`
39 `FROM orders_goods`
40 `WHERE order_id = 10012;`

Result Grid

Filter Rows:

Edit:

order_id	goods_id	quantity	price_out	is_shipped
10012	9999	4.000	9376	0
10012	10002	2.000	9376	0
10012	10003	2.000	9376	0
NULL	NULL	NULL	NULL	NULL

54 • `SELECT get_quantity_goods(10012,9999);`

Result Grid

Filter Rows:

Export:

get_quantity_goods(10012,9999)
4.000

Рис. 5.2.18 Результат роботи функції get_quantity_available_goods

5.2.19. Списання запчастини в замовлення, з використанням транзакції та курсору. Лістинг коду:

```
DELIMITER $$
DROP PROCEDURE IF EXISTS set_orders_goods4 $$
CREATE PROCEDURE set_orders_goods4(IN o_id INT, IN gd_id INT, IN gd_quantity
DECIMAL(7,3))

COMMENT 'Списання запчастини в замовлення (orders_goods) за goods_dscr_id,
перевірка наявності, є розподіл кількості по ціні'
LANGUAGE SQL
DETERMINISTIC
MODIFIES SQL DATA

BEGIN
    DECLARE message VARCHAR(200);
    DECLARE done INT DEFAULT 0;
    DECLARE var_goods_id INT;
    DECLARE var_quantity DECIMAL(7,3);
    DECLARE q DECIMAL(7,3);
    DECLARE old_quantity DECIMAL(7,3);
    DECLARE date_order DATE;

    -- Курсор для вибірки з блокуванням запису
    DECLARE cur CURSOR FOR SELECT goods_id, stock - reserve FROM goods WHERE
goods_dscr_id = gd_id FOR UPDATE;

    -- Обробка ситуації, коли курсор повертає NULL
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    -- Обробник помилок (ROLLBACK при SQL-винятках)
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SET message = 'Транзакцію скасовано через помилку';
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = message;
    END;

    START TRANSACTION;

    SELECT date_returned INTO date_order FROM orders WHERE order_id = o_id;
    IF date_returned THEN
        SET message = CONCAT('Помилка, orders_goods: замовлення з id = ',
o_id, ' закрито для редагування');
        ROLLBACK;
        -- Окремий ROLLBACK, якщо помилка ще до виконання курсора
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = message;
    END IF;

    -- перевіряємо доступну кількість запчастини
    SET q = get_quantity_available_goods(gd_id);
    IF q < gd_quantity THEN
        SET message = CONCAT('Помилка, orders_goods: запчастина з id = ',
gd_id, ' в такій кількості відсутня');
```



```

        ROLLBACK;
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = message;
    END IF;

    OPEN cur;

    -- Обробляємо кожний goods_id по черзі, цикл
    cycle: REPEAT
        FETCH cur INTO var_goods_id, var_quatity;

        -- чи вже є така запчастина в замовленні?
        SET old_quantity = get_quantity_goods(o_id, var_goods_id);

        CASE old_quantity
            WHEN 0 THEN
                -- Якщо немає, додаємо
                IF gd_quantity <= var_quatity THEN
                    INSERT INTO orders_goods (order_id, goods_id,
quantity)
                        VALUES (o_id, var_goods_id, gd_quantity);
                    LEAVE cycle;
                ELSE
                    INSERT INTO orders_goods (order_id, goods_id,
quantity)
                        VALUES (o_id, var_goods_id, var_quatity);
                    SET gd_quantity = gd_quantity - var_quatity;
                END IF;
            ELSE
                -- Якщо є, збільшуємо кількість
                IF gd_quantity <= var_quatity THEN
                    UPDATE orders_goods
                    SET quantity = gd_quantity + old_quantity
                    WHERE order_id = o_id AND goods_id =
var_goods_id;

                    LEAVE cycle;
                ELSE
                    UPDATE orders_goods
                    SET quantity = var_quatity + old_quantity
                    WHERE order_id = o_id AND goods_id =
var_goods_id;

                    SET gd_quantity = gd_quantity - var_quatity;
                END IF;
            END CASE;
        UNTIL done
        END REPEAT cycle;

    CLOSE cur;
    COMMIT;
END $$
DELIMITER ;

```

До виконання процедури

```

14 • SELECT goods_id, price_in,
15         stock, reserve
16 FROM goods
17 WHERE goods_dscr_id = 9999;

```

Result Grid | Filter Rows:

goods_id	price_in	stock	reserve
9999	7212	4.000	1.000
10002	7000	2.000	0.000
10003	7500	3.000	0.000
NULL	NULL	NULL	NULL

```

37 • SELECT order_id, goods_id, quantity,
38         price_out, is_shipped
39 FROM orders_goods
40 WHERE order_id = 10012;

```

Result Grid | Filter Rows: | Edit:

order_id	goods_id	quantity	price_out	is_shipped
10012	9999	1.000	9376	0
NULL	NULL	NULL	NULL	NULL

```

42 • SELECT order_id, date_returned, total FROM orders o WHERE o.order_id = 10012;

```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Cont

order_id	date_returned	total
10012	NULL	9376
NULL	NULL	NULL

Після виконання процедури `CALL set_orders_goods4(10012, 9999, 6);`

```

14 • SELECT goods_id, price_in,
15         stock, reserve
16 FROM goods
17 WHERE goods_dscr_id = 9999;

```

Result Grid | Filter Rows:

goods_id	price_in	stock	reserve
9999	7212	4.000	4.000
10002	7000	2.000	2.000
10003	7500	3.000	1.000
NULL	NULL	NULL	NULL

```

37 • SELECT order_id, goods_id, quantity,
38         price_out, is_shipped
39 FROM orders_goods
40 WHERE order_id = 10012;

```

Result Grid | Filter Rows: | Edit:

order_id	goods_id	quantity	price_out	is_shipped
10012	9999	4.000	9376	0
10012	10002	2.000	9376	0
10012	10003	1.000	9376	0
NULL	NULL	NULL	NULL	NULL

```

42 • SELECT order_id, date_returned, total FROM orders o WHERE o.order_id = 10012;

```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Cont

order_id	date_returned	total
10012	NULL	65632
NULL	NULL	NULL

Обробка помилок

`CALL set_orders_goods(10012, 9999, 10);`

✖	34	15:21:09	CALL set_orders_goods(10012, 9999, 10)	Error Code: 1644. Помилка, orders_goods: запчастини в такій кількості немає в наявності
✖	36	16:16:18	CALL set_orders_goods4(10020, 1, 1)	Error Code: 1644. Транзакцію скасовано через помилку

Рис. 5.2.19 Результат роботи процедури set_orders_goods4

5.2.20. Закриває order_repair сьогоднішньою датою. Лістинг коду функції:

```
DELIMITER $$
-- SET GLOBAL log_bin_trust_function_creators = 1 $$
DROP FUNCTION IF EXISTS func_order_repair $$
CREATE FUNCTION func_order_repair (o_id INT, e_id INT, price INT, dscr
VARCHAR(255))
RETURNS VARCHAR(255)
COMMENT 'закриває order_repair сьогоднішньою датою'
DETERMINISTIC
MODIFIES SQL DATA

BEGIN

    DECLARE message VARCHAR(200);
    DECLARE var_date DATE DEFAULT CURDATE();
    DECLARE date_order DATE;

    -- перевірка, чи замовлення не закрито для редагування
    SELECT date_returned INTO date_order FROM orders WHERE order_id = o_id;
    IF date_order THEN
        SET message = CONCAT('Помилка, orders_repair: замовлення з id = ',
o_id, ' закрито для редагування');
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = message;
    END IF;

    -- перевірка, чи була прийнята техніка в ремонт
    IF NOT EXISTS (SELECT 1 FROM orders_received WHERE order_id = o_id) THEN
        SET message = CONCAT('Помилка, orders_repair: замовлення з id = ',
o_id, ' немає техніки в ремонті');
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = message;
    END IF;

    CASE
    WHEN NOT EXISTS (SELECT 1 FROM orders_repair WHERE order_id = o_id AND
employee_id = e_id) THEN
        INSERT INTO orders_repair (order_id, employee_id, date_ready,
price_repair, description)
        VALUES (o_id, e_id, var_date, price, dscr);
    ELSE
        UPDATE orders_repair
        SET date_ready = var_date, price_repair = price, description = dscr
        WHERE order_id = o_id AND employee_id = e_id;
    END CASE;

    SET message = CONCAT('orders_repair: замовлення з id = ', o_id, '
успішно закрито');
    RETURN message;
END $$
DELIMITER ;
```

До виконання функції

```

11 • SELECT order_id, employee_id, price_repair, date_ready, description
12 FROM orders_repair WHERE order_id = 10019;

```

Result Grid			Filter Rows: <input type="text"/>	Edit:			Export/Import:	
order_id	employee_id	price_repair	date_ready	description				
10019	17	50000	2024-12-24	Заміна термоблоку, відновлення картриджу				

Після виконання функції

```

17 • SELECT func_order_repair(10019, 19, 71000, "Ремонт блока живлення");

```

Result Grid			Filter Rows: <input type="text"/>	Export:	Wrap Cell Content:
func_order_repair(10019, 19, 71000, "Ремонт блока живлення")					
orders_repair: замовлення з id = 10019 успішно закрито					

✓ 82 16:18:54 SELECT func_order_repair(10019, 19, 71000, "Ремонт блока живлення") 1 row(s) returned

```

11 • SELECT order_id, employee_id, price_repair, date_ready, description
12 FROM orders_repair WHERE order_id = 10019;

```

Result Grid			Filter Rows: <input type="text"/>	Edit:			Export/Import:	
order_id	employee_id	price_repair	date_ready	description				
10019	17	50000	2024-12-24	Заміна термоблоку, відновлення картриджу				
10019	19	71000	2024-12-25	Ремонт блока живлення				
NULL	NULL	NULL	NULL	NULL				

Рис. 5.2.20.1 Результат роботи функції func_order_repair

Лістинг коду процедури:

```

DELIMITER $$
DROP PROCEDURE IF EXISTS set_orders_repair $$
CREATE PROCEDURE set_orders_repair(IN o_id INT, IN e_id INT, IN price INT, IN
descr VARCHAR(255))

COMMENT 'апдейт orders_repair по закінченню ремонту, дата - сьогодні,
price_repair, description'
LANGUAGE SQL
DETERMINISTIC
MODIFIES SQL DATA

BEGIN
    DECLARE message VARCHAR(200);
    DECLARE var_date DATE DEFAULT CURDATE();
    DECLARE date_order DATE;

    -- перевірка, чи замовлення не закрито для редагування
    SELECT date_returned INTO date_order FROM orders WHERE order_id = o_id;
    IF date_order THEN
        SET message = CONCAT('Помилка, orders_repair: замовлення з id = ',
o_id, ' закрито для редагування');

```

```

        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = message;
    END IF;

    -- перевірка, чи була прийнята техніка в ремонт
    IF NOT EXISTS (SELECT 1 FROM orders_received WHERE order_id = o_id) THEN
        SET message = CONCAT('Помилка, orders_repair: замовлення з id = ',
            o_id, ' немає техніки в ремонті');
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = message;
    END IF;

    IF NOT EXISTS (SELECT 1 FROM orders_repair WHERE order_id = o_id AND
        employee_id = e_id) THEN
        INSERT INTO orders_repair (order_id, employee_id, date_ready,
            price_repair, description)
            VALUES (o_id, e_id, var_date, price, dscr);
    ELSE
        UPDATE orders_repair
            SET date_ready = var_date, price_repair = price, description = dscr
            WHERE order_id = o_id AND employee_id = e_id;
    END IF;
END $$
DELIMITER ;

```

```
11 • SELECT * FROM orders_repair WHERE order_id = 10020;
```

Result Grid Filter Rows: <input type="text"/> Edit: Export					
order_id	employee_id	date_ready	price_repair	description	
10020	15	NULL	0	NULL	
NULL	NULL	NULL	NULL	NULL	

✓ 25 15:49:53 CALL set_orders_repair(10020, 15, 75000, 'Заміна картриджу') 1 row(s) affected 0.016 sec

order_id	employee_id	date_ready	price_repair	description	
10020	15	2024-12-26	75000	Заміна картриджу	
NULL	NULL	NULL	NULL	NULL	

✓ 27 15:52:13 CALL set_orders_repair(10020, 5, 25000, 'Заміна шестерні подачі паперу') 1 row(s) affected 0.000 sec

order_id	employee_id	date_ready	price_repair	description	
10020	5	2024-12-26	25000	Заміна шестерні подачі паперу	
10020	15	2024-12-26	75000	Заміна картриджу	
NULL	NULL	NULL	NULL	NULL	

Рис. 5.2.20.2 Результат роботи процедури set_orders_repair

5.2.21. Ставити дату відвантаження сьогодні за order_id та «відвантажено» в orders_goods на все goods_id, з використанням **двох курсорів**. Лістинг коду:

```

DELIMITER $$
DROP PROCEDURE IF EXISTS set_date_orders $$
CREATE PROCEDURE set_date_orders (IN o_id INT)

```



```

COMMENT 'Ставимо дату замовлення - сьогодні за order_id та «відвантажено» в
orders_goods на всі goods_id'
LANGUAGE SQL
DETERMINISTIC
MODIFIES SQL DATA

BEGIN
    DECLARE message VARCHAR(200);
    DECLARE var_shipped INT;
    DECLARE var_goods_id INT;
    DECLARE var_date DATE;
    DECLARE done INT DEFAULT 0;

    -- Курсор для витягу ремонтів з orders_repair
    DECLARE cur_orp CURSOR FOR SELECT date_ready FROM orders_repair WHERE
order_id = o_id;

    -- Курсор для витягу запчастин з orders_goods
    DECLARE cur_og CURSOR FOR SELECT goods_id, is_shipped FROM orders_goods
WHERE order_id = o_id;

    -- Обробка ситуації, коли курсор повертає NULL
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    IF EXISTS (SELECT date_returned FROM orders WHERE order_id = o_id) THEN
        SET message = CONCAT('Помилка, orders_goods: замовлення з id = ',
o_id, ' вже закрито для редагування');
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = message;
    END IF;

    -- перевірка, чи була прийнята техніка в ремонт, та вже відремонтована
    IF EXISTS (SELECT 1 FROM orders_received WHERE order_id = o_id) THEN
        IF (SELECT COUNT(*) FROM orders_repair WHERE order_id = o_id) > 0
THEN
            OPEN cur_orp;
            REPEAT
                FETCH cur_orp INTO var_date;
                IF var_date IS NULL THEN
                    SET message = CONCAT('Помилка, orders_repair:
замовлення з id = ', o_id, ' техніка ще в ремонті, не можу закрити
замовлення');
                    CLOSE cur_orp;
                    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = message;
                END IF;
            UNTIL done
            END REPEAT;
            CLOSE cur_orp;
            SET done = 0;
        ELSE
            SET message = CONCAT('Помилка, orders_repair: замовлення з
id = ', o_id, ' техніка ще в ремонті, не можу закрити замовлення');
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = message;
        END IF;
    END IF;

    -- перевірка, чи були зарезервовані запчастини в orders_goods, якщо
були, то is_shipped = 1
    IF (SELECT COUNT(*) FROM orders_goods WHERE order_id = o_id) > 0 THEN
        OPEN cur_og;
        WHILE done = 0 DO
            FETCH cur_og INTO var_goods_id, var_shipped;
            IF var_shipped = 0 THEN
                UPDATE orders_goods
                SET is_shipped = 1

```

```

        WHERE order_id = o_id AND goods_id = var_goods_id;
    END IF;
END WHILE;
CLOSE cur_og;
END IF;

-- проставляємо дату замовлення
UPDATE orders SET date_returned = CURDATE() WHERE order_id = o_id;

END $$
DELIMITER ;

```

До

11 • SELECT * FROM orders_goods WHERE order_id = 10020;

order_id	goods_id	quantity	price_out	descriptio	is_shipped
10020	1	1.000	8722	NULL	0
10020	2	1.000	1880	NULL	0
NULL	NULL	NULL	NULL	NULL	NULL

12 • SELECT * FROM orders_repair WHERE order_id = 10020;

order_id	employee_id	date_ready	price_repair	description
10020	5	2024-12-26	25500	Заміна шестерні подачі паперу
10020	15	2024-12-26	75000	Заміна картриджу
NULL	NULL	NULL	NULL	NULL

13 • SELECT order_id, date_returned, total FROM orders WHERE order_id = 10020;

order_id	date_returned	total
10020	NULL	111102
NULL	NULL	NULL

✓ 61 16:50:38 CALL set_date_orders(10020) 1 row(s) affected

Після

11 • SELECT * FROM orders_goods WHERE order_id = 10020;

order_id	goods_id	quantity	price_out	descriptio	is_shipped
10020	1	1.000	8722	NULL	1
10020	2	1.000	1880	NULL	1
NULL	NULL	NULL	NULL	NULL	NULL

13 • `SELECT order_id, date_returned, total FROM orders WHERE order_id = 10020;`

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell

order_id	date_returned	total
10020	2024-12-26	111102
NULL	NULL	NULL

65 16:52:33 CALL set_date_orders(10020) Error Code: 1644. Помилка, orders_goods: замовлення з id = 10020 вже закрито для редагування

Рис. 5.2.21 Результат роботи процедури set_date_orders

5.2.22. Повертає відомість за активними постачальниками. Лістинг коду:

```
DELIMITER $$
DROP PROCEDURE IF EXISTS get_customer_report $$
CREATE PROCEDURE get_supplier_report()

COMMENT 'Відомість за активними постачальниками'
LANGUAGE SQL
DETERMINISTIC
READS SQL DATA

BEGIN

    SELECT a.customer_id,
           MAX(balance) as start_balance,
           SUM(total) as arrival_total,
           SUM(pay_sum) as pay_total,
           MAX(balance) - SUM(total) + SUM(pay_sum) as total_balance
    FROM arrivals a
    LEFT JOIN pay p ON a.arrival_id = p.arrival_id
    JOIN customers c ON c.customer_id = a.customer_id
    GROUP BY customer_id;

END $$
DELIMITER ;
```

69 • `CALL get_supplier_report();`

Result Grid | Filter Rows: | Export: | Wrap Cell Cont

customer_id	start_balance	arrival_total	pay_total	total_balance
266	0	1913946	1338885	-575061
7567	0	274345	274345	0
6685	0	451263	451263	0

66 16:59:23 CALL get_supplier_report() 949 row(s) returned 0.016 sec / 0.000 sec

Рис. 5.2.22.1 Результат роботи процедури get_supplier_report

Подивимось, чи потрібна оптимізація процедури, використаємо EXPLAIN.


```

99 • SELECT a.customer_id,
100         MAX(balance) as start_balance,
101         SUM(total) as arrival_total,
102         SUM(pay_sum) as pay_total,
103         MAX(balance) - SUM(total) + SUM(pay_sum) as total_balance
104 FROM arrivals a
105 JOIN pay p ON a.arrival_id = p.arrival_id
106 JOIN customers c ON c.customer_id = a.customer_id
107 GROUP BY customer_id;

```

Tabular Explain											
id	select_type	table	partiti...	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	a		ALL	PRIMARY,fk_arrivals_customers1_idx				936	100.00	Using temporary
1	SIMPLE	c		eq_ref	PRIMARY	PRIMARY	4	ss_innodb.a.customer...	1	100.00	
1	SIMPLE	p		ref	fk_pay_arrivals1_idx	fk_pay_arrivals1_idx	5	ss_innodb.a.arrival_id	10	100.00	

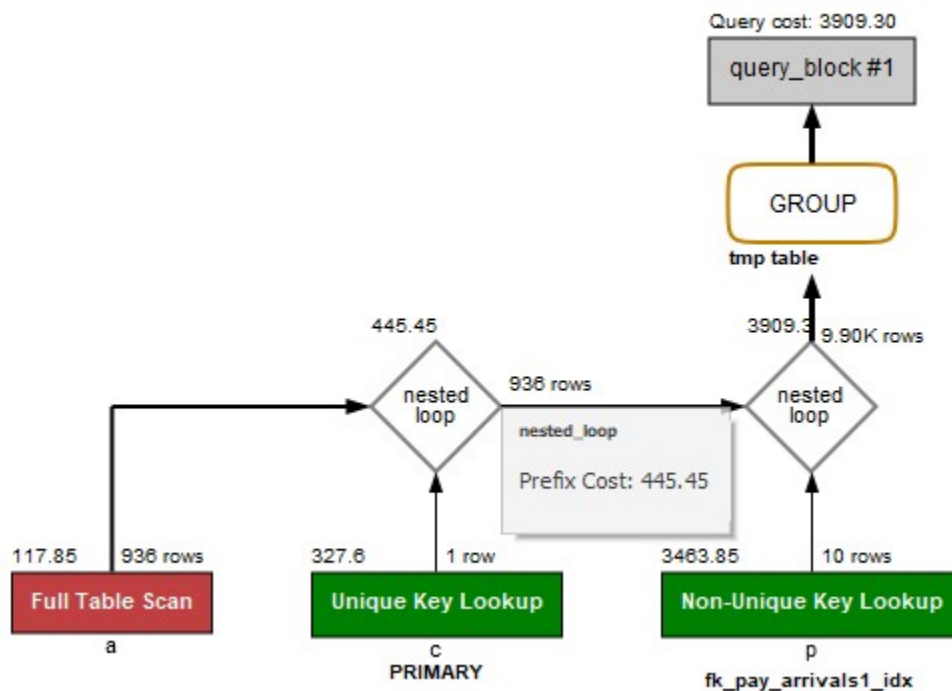


Рис. 5.2.22.2 Скриншот EXPLAIN запиту, що використовує процедура

Тут хоча і використовується Full Table Scan, але тільки обраних рядків (936), які і повертає процедура в результаті, а в решті використовується пошук за індексованими полями.

5.2.23. Повертає баланс постачальника на вказану дату. Лістинг коду:

```

DELIMITER $$
-- SET GLOBAL log_bin_trust_function_creators = 1 $$
DROP FUNCTION IF EXISTS get_supplier_card_start $$
CREATE FUNCTION get_supplier_card_start (id INT, var_date DATE)
RETURNS INT
COMMENT 'повертає баланс постачальника на вказану дату за його customer_id'
DETERMINISTIC
READS SQL DATA

BEGIN
    DECLARE total_balance INT DEFAULT 0;

    SELECT balance + SUM(pay_sum) - SUM(total) INTO total_balance

```

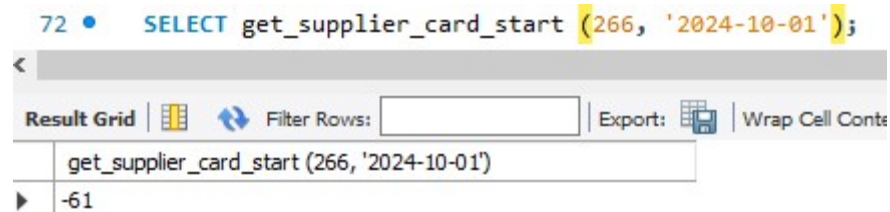
```

FROM arrivals a
LEFT JOIN pay p ON a.arrival_id = p.arrival_id
JOIN customers c ON c.customer_id = a.customer_id
WHERE a.customer_id = id and date_arrival <= var_date
GROUP BY a.customer_id;

RETURN total_balance;
END $$
DELIMITER ;

```

72 • SELECT get_supplier_card_start (266, '2024-10-01');



get_supplier_card_start (266, '2024-10-01')
-61

Рис. 5.2.23 Результат роботи функції get_supplier_card_start

5.2.24. Повертає Картку постачальника. Лістинг коду:

```

DELIMITER $$
DROP PROCEDURE IF EXISTS get_supplier_card $$
CREATE PROCEDURE get_supplier_card(IN id INT, IN date_start DATE, IN date_end DATE)

COMMENT 'Картка постачальника за customer_id'
LANGUAGE SQL
DETERMINISTIC
READS SQL DATA

BEGIN

SELECT a.customer_id,
       get_supplier_card_start(id, date_start) as start_balance,
       date_arrival,
       total as arrival_total,
       date_pay,
       pay_sum as pay_total,
       get_supplier_card_start(id, date_start) + coalesce(pay_sum, 0) -
total as total_balance
FROM arrivals a
LEFT JOIN pay p ON a.arrival_id = p.arrival_id
WHERE a.customer_id = id and (date_arrival BETWEEN date_start AND
date_end);

END $$
DELIMITER ;

```

75 • `CALL get_supplier_card(266, '2024-01-01', '2024-12-31');`

customer_id	start_balance	date_arrival	arrival_total	date_pay	pay_total	total_balance
266	0	2024-01-01	894161	2024-01-09	894100	-61
266	0	2024-03-24	444785	2024-02-17	444785	0
266	0	2024-11-05	315000	NULL	NULL	-315000
266	0	2024-11-06	260000	NULL	NULL	-260000

75 • `CALL get_supplier_card(266, '2024-10-01', '2024-12-31');`

customer_id	start_balance	date_arrival	arrival_total	date_pay	pay_total	total_balance
266	-61	2024-11-05	315000	NULL	NULL	-315061
266	-61	2024-11-06	260000	NULL	NULL	-260061

Рис. 5.2.24 Результат роботи процедури

Висновки:

- написання функцій і процедур значно спрощує код і дає можливість просто його перевикористовувати;
- використання не великих проміжних функцій полегшує зв'язування таблиць, а де коли взагалі без нього вдається обійтися, в той же час код функцій пишеться оптимальним образом, використовуються індексовані поля, а найчастіше взагалі первинний ключ в конструкції WHERE, тому оптимізація функцій у більшості випадків не потрібна;
- створене в попередніх роботах поле total в таблиці orders, де знаходиться сума замовлення суттєво спрощує написання коду, та багаторазово збільшує швидкодію запита, що оброблює всі замовлення за сумою, при кількості рядків в таблиці до 10 тисяч;
- порівняли результат двох однакових процедур, одна з типом даних INT проти DECIMAL(10,2) у іншій. Є невеликий приріст швидкості у першій, десь 10%.
- навчились створювати функції і процедури, що додають, та модифікують дані в таблицях, використали цикли та умовні оператори, а також використали курсор (навіть два курсори в одній процедурі) та тимчасові таблиці, спробував використати транзакцію, для списання

запчастин в замовленні, щоб бути впевненим, що операція завершена коректно, або не завершена взагалі, що важливо для високонавантажених баз даних.