

МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
Харківський національний університет радіоелектроніки

Кафедра системотехніки

ЗВІТ

з виконання завдань практичного заняття № 7
дисципліни «Проектування високонавантажених систем
зберігання даних»

на тему: «СТВОРЕННЯ ТРАНЗАКЦІЙ 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 з розробки транзакцій, що використовуються в збережених процедурах і функціях, для забезпечення основних бізнес процесів високонавантаженої інформаційної системи;
- формування необхідних практичних умінь для створення транзакцій, з урахуванням особливостей роботи високонавантаженої інформаційної системи зберігання даних.

Завдання на самостійну роботу

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

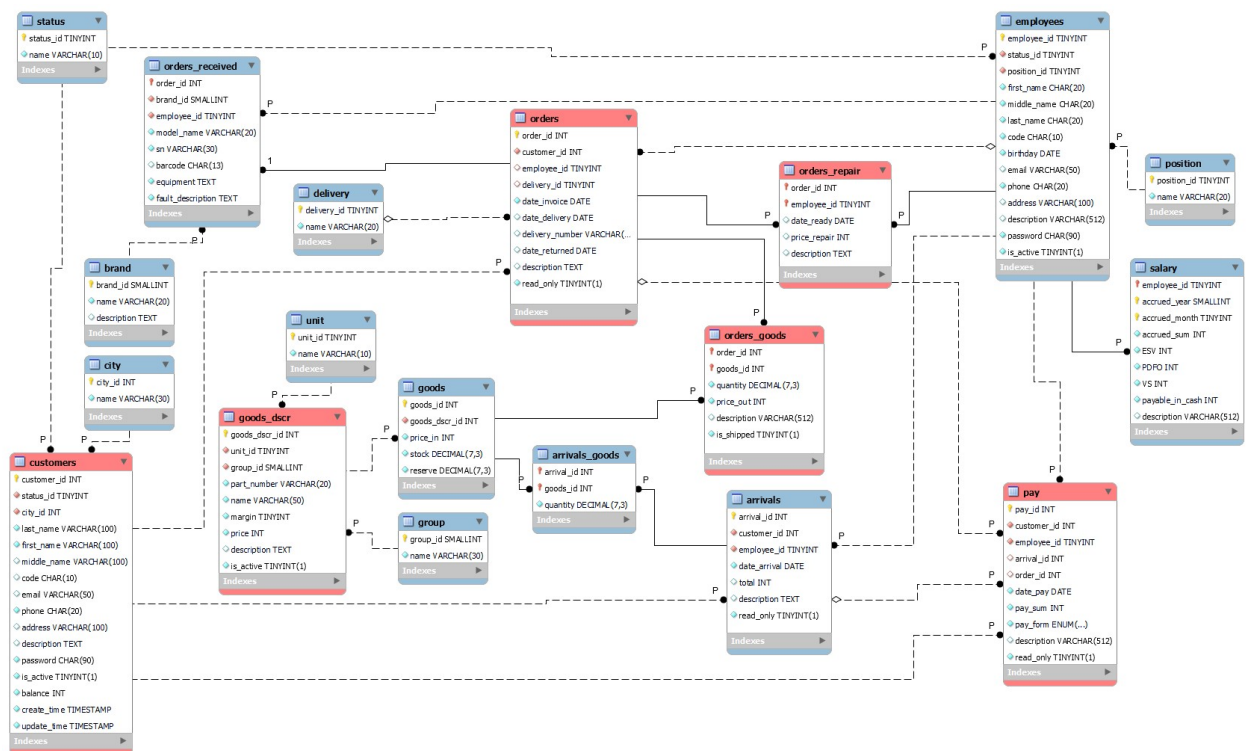


Рис. 7.1. Скріншот схеми фізичної моделі бази даних з таблицями типу InnoDB

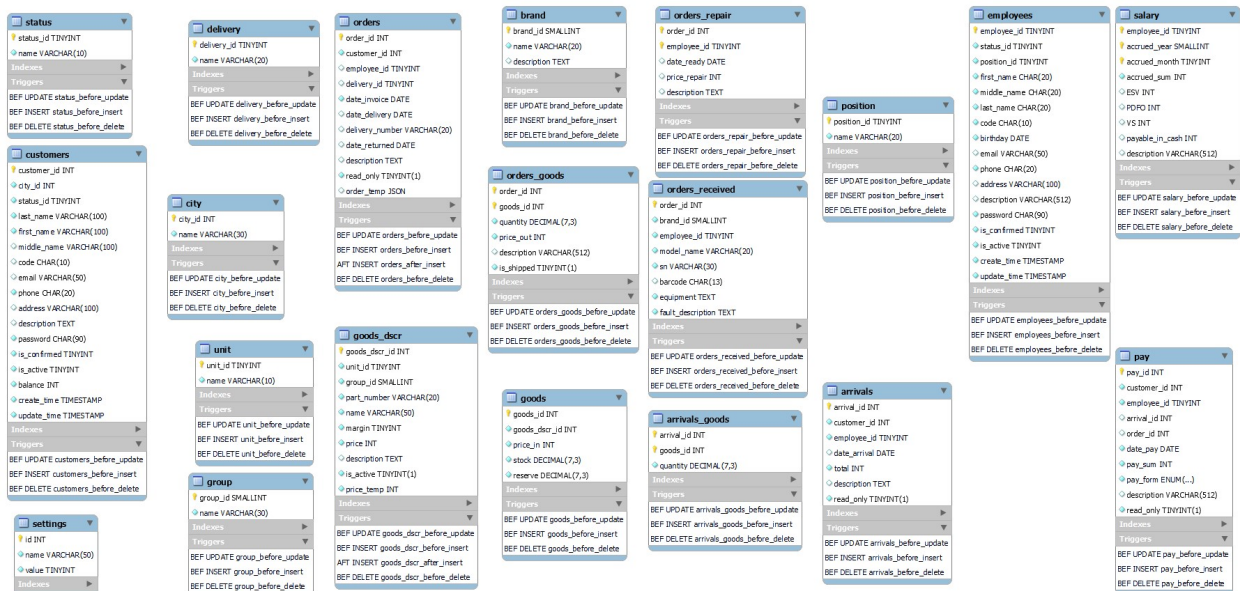


Рис. 7.2. Скріншот схеми фізичної моделі бази даних з таблицями типу MyIsam

Завдання 7.1. Для бази даних з таблицями MyIsam, розробленої відповідно до завдання 1.2 практичного заняття № 1, розробити збережену процедуру. Збережена процедура має забезпечувати виконання одного з SQL-запитів завдання 2.2 (п.2.3.9) із блокуванням використовуваних таблиць.

Перед виконанням завдання протестувати доступ до заблокованої базової таблиці, створивши два підключення до сервера MySQL (під час тестування використовувати два блокування таблиці – з атрибутами READ і WRITE).

Є процедура set_order, яка вставляє дані одразу в чотири таблиці, створюючи новий ордер:

```
DELIMITER $$
DROP PROCEDURE IF EXISTS set_order $$
CREATE PROCEDURE set_order(
    IN order_id INT,
    IN customer_id INT,
    IN description VARCHAR(255),
    IN brand_id SMALLINT,
    IN rc_employee_id TINYINT,
    IN model_name VARCHAR(20),
    IN sn VARCHAR(30),
    IN equipment VARCHAR(200),
    IN fault_description VARCHAR(200),
    IN rp_employee_id TINYINT,
    IN goods_id INT,
    IN quantity decimal(7,3)
)

```

```

COMMENT ' Створення нового замовлення, запис у 4 таблиці в одній транзакції '
LANGUAGE SQL
DETERMINISTIC
MODIFIES SQL DATA

BEGIN

    DECLARE new_order_id INT;
    DECLARE order_date DATE;
    DECLARE message VARCHAR(200);

    SET new_order_id = order_id;
    SET order_date = CURDATE();

    -- записуємо дані в 4 таблиці
    IF order_id IS NULL THEN
        INSERT INTO orders(customer_id, date_invoice, description)
        VALUE(customer_id, order_date, description);

        SET new_order_id = LAST_INSERT_ID();
    END IF;

    IF brand_id THEN
        INSERT INTO orders_received(order_id, brand_id, employee_id, model_name,
sn, equipment, fault_description)
        VALUES (new_order_id, brand_id, rc_employee_id, model_name, sn,
equipment, fault_description);
    END IF;

    IF rp_employee_id THEN
        INSERT INTO orders_repair(order_id, employee_id)
        VALUES (new_order_id, rp_employee_id);
    END IF;

    IF goods_id THEN
        INSERT INTO orders_goods(order_id, goods_id, quantity)
        VALUES (new_order_id, goods_id, quantity);
    END IF;

    -- CALL set_order(null, 1, "Клієнт просить перевірити роботу на великих тиражах", 3, 5,
"Р3005" , "1233412312", "без кабелю живлення", "поганий друк", 7, 1, 2);

END $$
DELIMITER ;

```

Є два підключення до БД (рис. 7.3):

MySQL Connections

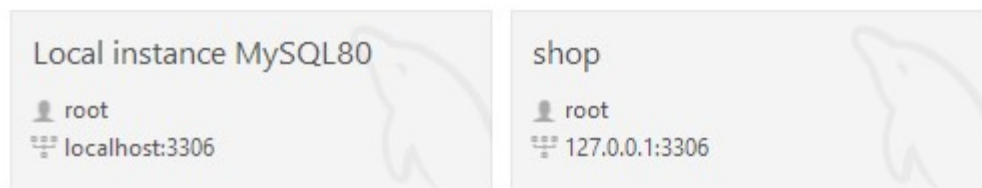


Рис. 7.3 Скріншот підключень до БД MySQL

Заблокуємо таблиці для читання, та перевіримо можливість читати та писати в БД з обох підключень(рис. 7.4):

```
LOCK TABLES orders READ, orders_received READ, orders_repair READ, orders_goods READ;
```

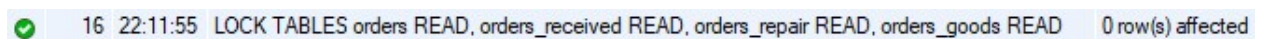
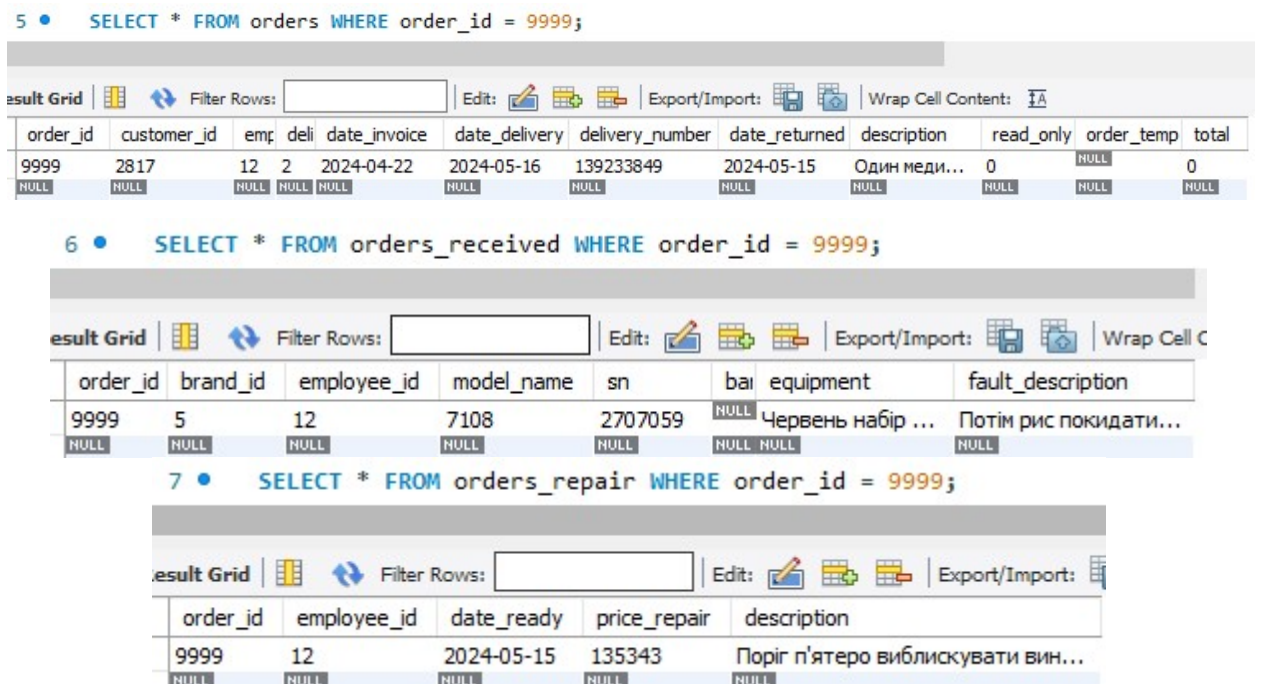


Рис. 7.4 Скріншот запиту блокування таблиць

Виконуємо читання з таблиць(рис. 7.5):

```
SELECT * FROM orders WHERE order_id = 9999;  
SELECT * FROM orders_received WHERE order_id = 9999;  
SELECT * FROM orders_repair WHERE order_id = 9999;  
SELECT * FROM orders_goods WHERE order_id = 9999;
```



8 • `SELECT * FROM orders_goods WHERE order_id = 9999;`

order_id	goods_id	quantity	price_out	description	is_shipped
9999	2794	1.009	5204	Повністю відпові...	1

Рис. 7.5 Скріншот запиту зчитування даних з таблиць

Зчитуємо з іншого облікового запису «shop» (рис.7.6):

5 • `SELECT * FROM orders WHERE order_id = 9999;`

order_id	customer_id	employee_id	delivery_id	date_invoice	date_delivery	delivery_number	date_returned	description	read_only	order_temp	total
9999	2817	12	2	2024-04-22	2024-05-16	139233849	2024-05-15	Один ме...	0	NULL	0

6 • `SELECT * FROM orders_received WHERE order_id = 9999;`

order_id	brand_id	employee_id	model_name	sn	barcode	equipment	fault_description
9999	5	12	7108	2707059	NULL	Червень н...	Потім рис покид...

7 • `SELECT * FROM orders_repair WHERE order_id = 9999;`

order_id	employee_id	date_ready	price_repair	description
9999	12	2024-05-15	135343	Поріг п'ятеро вибли...

8 • `SELECT * FROM orders_goods WHERE order_id = 9999;`

order_id	goods_id	quantity	price_out	description	is_shipped
9999	2794	1.009	5204	Повністю відп...	1

Рис. 7.6 Скріншот запиту зчитування даних з таблиць з іншого облікового запису «shop»

Тепер спробуємо записати(рис. 7.7-7.8):

`CALL set_order(null, 1, "Клієнт просить перевірити роботу на великих тиражах", 3, 5, "P3005", "1233412312", "без кабелю живлення", "поганий друк", 7, 1, 2);`

21 22:25:09 `CALL set_order(null, 1, "Клієнт просить перевірити роботу на великих тиражах", 3, 5, "P300...` Error Code: 1100. Table 'c' was not locked with LOCK TABLES

Рис. 7.7 Скріншот спроби виконати запит запису даних (через процедуру `set_order`)

14 22:32:45 CALL set_order(null, 1, "Клієнт просить перевірити роботу на ДУЖЕ великих тиражах", 3, 6, "3005", "12334-3... Error Code: 2013. Lost connection to MySQL server during query 30.000 sec

Рис. 7.8 Скріншот спроби виконати запит запису даних (через процедуру set_order) з іншого облікового запису «shop»

Заблокуємо таблиці для запису(рис. 7.9-7.10):

LOCK TABLES orders WRITE, orders_received WRITE, orders_repair WRITE, orders_goods WRITE;

23 22:37:03 CALL set_order(null, 1, "Клієнт просить перевірити роботу на великих тиражах", 3, 5, "P300... 1 row(s) affected

Рис. 7.9 Скріншот виконання процедури set_order запису даних

5 • SELECT * FROM orders ORDER BY order_id DESC LIMIT 1;

order_id	customer_id	emp_id	deli	date_invoice	date_delivery	delivery_num	date_returned	description	read_only	order_temp	total
10025	1	NULL	NULL	2025-01-01	NULL	NULL	NULL	Клієнт просить перевірит...	0	NULL	0
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

6 • SELECT * FROM orders_received ORDER BY order_id DESC LIMIT 1;

order_id	brand_id	employee_id	model_name	sn	bai	equipment	fault_description
10025	3	5	P3005	1233412312	NULL	без кабелю живле...	поганий друк
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

7 • SELECT * FROM orders_repair ORDER BY order_id DESC LIMIT 1;

order_id	employee_id	date_ready	price_repair	description
10025	7	NULL	0	NULL
NULL	NULL	NULL	NULL	NULL

8 • SELECT * FROM orders_goods ORDER BY order_id DESC LIMIT 1;

order_id	goods_id	quantity	price_out	description	is_shipped
10025	1	2.000	8722	NULL	0
NULL	NULL	NULL	NULL	NULL	NULL

Рис. 7.10 Скріншот перевірки, що записалось в таблиці

Спробуємо зчитати дані, а потім вставити нові з іншого облікового запису «shop» (рис. 7.11):

Рис. 7.11 Скріншот помилки при спробі прочитати дані вже на першій таблиці

Зробимо **UNLOCK TABLES**, та запишемо дані з іншого облікового запису «shop» (рис. 7.12):

28 22:51:53 UNLOCK TABLES

16 22:52:43 CALL set_order(null, 1, "Клієнт просить перевірити роботу на ДУЖЕ великих тиражах", 3, 6, "3005", "12334-3... 1 row(s) affected

5 • SELECT * FROM orders ORDER BY order_id DESC LIMIT 1;

order_id	customer_id	employee_id	delivery_id	date_invoice	date_delivery	delivery_number	date_returned	description	read_only	order_temp	total
10026	1	NULL	NULL	2025-01-01	NULL	NULL	NULL	Клієнт п...	0	NULL	0
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

6 • SELECT * FROM orders_received ORDER BY order_id DESC LIMIT 1;

order_id	brand_id	employee_id	model_name	sn	barcode	equipment	fault_description
10026	3	6	3005	12334-3005	NULL	з кабелем ...	дуже поганий д...
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

7 • SELECT * FROM orders_repair ORDER BY order_id DESC LIMIT 1;

order_id	employee_id	date_repair	price_repair	description
10026	6	NULL	0	NULL
NULL	NULL	NULL	NULL	NULL

8 • SELECT * FROM orders_goods ORDER BY order_id DESC LIMIT 1;

order_id	goods_id	quantity	price_out	description	is_shipped
10026	6	2.000	9626	NULL	0
NULL	NULL	NULL	NULL	NULL	NULL

Рис. 7.12 Скріншот розблокування таблиць та вставки даних з іншого облікового запису «shop» , а також перевірки, що записалось.

Таким чином, переконалися, що блокування таблиць унеможливилює запис (в режимі READ) для всіх сесій, або читання та запис у режимі WRITE з інших облікових записів. Також дізналися, що блокування на читання, або запис з іншого облікового запису є тимчасовим, тобто СУБД чекає певний час на знаття блокування і якщо дочекалася, то здійснює читання або запис одразу після зняття блокування, якщо ні, то через 30 секунд з'являється помилка.

Завдання 7.2 Для бази даних з таблицями InnoDB, розробленої відповідно до завдання 1.3 практичного заняття № 1, розробити перелік транзакцій, необхідних для забезпечення бізнес-функцій, на які впливає специфіка високонавантажених систем. Під час створення переліку рахувати результати виконання п.2.3.9 завдання 2.2.

Таблиця 7.1 – Транзакції для високонавантаженої системи

№	Призначення транзакції	Найменування таблиць
1	Додавання нового товару: – занесення й одержання ID найменування товару; – додавання нової ціни товару;	goods_dscr goods
2	Приход товару: – занесення й одержання ID приходу; – занесення інформації по товарам в опис приходу; – додавання нової ціни товару, якщо потрібно; – перерахунок складу (тригер);	arrivals arrivals_goods goods
3	Оформлення замовлення на товар: – занесення й одержання ID замовлення; – занесення інформації про надходження техніки ремонт; – розподіл техніки по майстрам, що ремонтують; – занесення інформації в кошик; – перерахунок складу (тригер);	orders orders_received orders_repair orders_goods goods

Завдання 7.3. Відповідно до переліку завдання 7.2, розробити транзакції для бази даних з таблицями типу InnoDB і SQL-запити до них. Розроблені транзакції мають задовольняти принципи ACID. Провести:
– тестування розроблених транзакцій;
– контроль виконаних транзакцій за допомогою бінарного журналу СУБД. MySQL.

7.3.1 Додавання нового товару. Лістинг коду процедури set_goods_dscr:

```
DELIMITER $$
DROP PROCEDURE IF EXISTS set_goods_dscr $$
CREATE PROCEDURE set_goods_dscr(
    IN unit_id tinyint,
    IN group_id smallint,
    IN part_number CHAR(10),
    IN name VARCHAR(50),
    IN margin tinyint,
```

```

IN new_price int,
IN description VARCHAR(255),
IN price_in INT
)

COMMENT 'Створення нового найменування товару, запис у 2 таблиці в одній транзакції '
-- CALL set_goods_dscr(1, 1, 234WRV6783, "Барабан P3005" , 50, null, "long life", 3500);
LANGUAGE SQL
DETERMINISTIC
MODIFIES SQL DATA

BEGIN

DECLARE new_goods_dscr_id INT;
DECLARE new_goods_id INT;

IF new_price IS NULL THEN
    SET new_price = ROUND(price_in * (1 + margin / 100));
END IF;

START TRANSACTION;

    INSERT INTO goods_dscr(unit_id, group_id, part_number, name, margin, price,
description)
    VALUE(unit_id, group_id, part_number, name, margin, new_price, description);
    SET new_goods_dscr_id = LAST_INSERT_ID();

    INSERT INTO goods(goods_dscr_id, price_in)
    VALUES (new_goods_dscr_id, price_in);

COMMIT;

END $$
DELIMITER ;

```

7 21:20:23 CALL set_goods_dscr(1, 1, '234WRV6783', "Барабан P3005" , 50, null, "long life", 3500) 0 row(s) affected 0.047 sec

5 • SELECT * FROM goods_dscr ORDER BY goods_dscr_id DESC LIMIT 1;

goods_dscr_id	unit_id	group_id	part_number	name	margin	price	description
10005	1	1	234WRV6783	Барабан P3005	50	5250	long life
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

6 • SELECT * FROM goods ORDER BY goods_id DESC LIMIT 1;

goods_id	goods_dscr_id	price_in	stock	reserve
10004	10005	3500	0.000	0.000
NULL	NULL	NULL	NULL	NULL

Рис. 7.13 Скріншот виконання та результати процедури set_goods_dscr

Визвемо процедуру з параметром, що викличе помилку при вставці в другу таблицю(рис. 7.14):

10 21:29:46 CALL set_goods_dscr(1, 1, '234WRV6783', 'Барабан P3005', 50, null, 'long life', -3500) Error Code: 1264. Out of range value for column 'price' at row 1 0.000 sec

5 • `SELECT * FROM goods_dscr ORDER BY goods_dscr_id DESC LIMIT 2;`

goods_dscr_id	unit_id	group_id	part_number	name	margin	price	description
10005	1	1	234WRV6783	Барабан P3005	50	5250	long life
10004	2	5	WTR2T2RC89	Тавар RD022	30	25000	Дівка синок століття
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

6 • `SELECT * FROM goods ORDER BY goods_id DESC LIMIT 2;`

goods_id	goods_dscr_id	price_in	stock	reserve
10004	10005	3500	0.000	0.000
10003	9999	7500	3.000	2.000
NULL	NULL	NULL	NULL	NULL

Рис. 7.14. Скріншот виконання та результати процедури set_goods_dscr при появі помилки

Нових записів в таблицях не з'явилося, що свідчить о том, що транзакція відкотилась вірно.

7.3.2. Приход товару. Лістинг коду процедури set_arrival та скріншоти виконання (рис. 7.15):

```
DELIMITER $$
DROP PROCEDURE IF EXISTS set_arrival $$
CREATE PROCEDURE set_arrival(
    IN arrival_id INT,
    IN customer_id INT,
    IN employee_id tinyint,
    IN total INT,
    IN description VARCHAR(255),

    IN new_goods_dscr_id INT,
    IN price INT,
    IN quantity decimal(7,3)
)
```

```
COMMENT ' Створення нового приходу, запис у 3 таблиці в одній транзакції '
-- CALL set_arrival(null, 1, 1, 300000, "якийсь опис", 10005, 3600, 10);
```

```
LANGUAGE SQL
```

DETERMINISTIC
MODIFIES SQL DATA

BEGIN

```
DECLARE new_arrival_id INT;  
DECLARE new_goods_id INT;  
DECLARE message VARCHAR(200);
```

```
SET new_arrival_id = arrival_id;
```

```
START TRANSACTION;
```

```
IF arrival_id IS NULL THEN
```

```
    INSERT INTO arrivals(customer_id, employee_id, total, description)  
    VALUE(customer_id, employee_id, total, description);
```

```
    SET new_arrival_id = LAST_INSERT_ID();  
END IF;
```

```
IF new_goods_dscr_id IS NULL THEN
```

```
    SET message = CONCAT('Помилка, goods_dscr: треба вказати id  
запчастини');
```

```
    ROLLBACK;
```

```
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = message;
```

```
ELSE
```

```
    IF EXISTS (SELECT 1 FROM goods_dscr WHERE goods_dscr_id =  
new_goods_dscr_id ) THEN
```

```
        -- перевіряємо, чи є товар з потрібною ціною, якщо ні, додаємо ціну
```

```
        SELECT goods_id FROM goods WHERE goods_dscr_id =  
new_goods_dscr_id AND price_in = price INTO new_goods_id;
```

```
        IF new_goods_id IS NULL THEN
```

```
            INSERT INTO goods(goods_dscr_id, price_in)
```

```
            VALUES (new_goods_dscr_id, price);
```

```
            SET new_goods_id = LAST_INSERT_ID();
```

```
        END IF;
```

```
        -- додаємо запис в прихід
```

```
        INSERT INTO arrivals_goods(arrival_id, goods_id, quantity)
```

```
        VALUES (new_arrival_id, new_goods_id, quantity);
```

```
    ELSE
```

```
        SET message = CONCAT('Помилка, goods_dscr: запчастина з id = ',  
new_goods_dscr_id, ' не існує');
```

```
        ROLLBACK;
```

```
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = message;
```

```
    END IF;
```

```
END IF;
```

```
COMMIT;
```

```
END $$
```

```
DELIMITER ;
```

24 21:48:00 CALL set_arrival(null, 1, 1, 300000, "якийсь опис", 10005, 3600, 10) 0 row(s) affected 0.032 sec

7 •

SELECT * FROM arrivals ORDER BY arrival_id DESC LIMIT 2;

result Grid

Filter Rows:

Edit:

Export/Import

arrival_id	customer_id	employee_id	date_arrival	total	description
1009	1	1	NULL	300000	якийсь опис
1003	266	1	2024-11-06	260000	NULL
NULL	NULL	NULL	NULL	NULL	NULL

8 •

SELECT *

9 FROM arrivals_goods

10 ORDER BY arrival_id

11 DESC LIMIT 2;

result Grid

Filter Rows:

arrival_id	goods_id	quantity
1009	10005	10.000
1003	6	20.000
NULL	NULL	NULL

6 •

SELECT * FROM goods

7 ORDER BY goods_id DESC

8 LIMIT 2;

result Grid

Filter Rows:

goods_id	goods_dscr_id	price_in	stock
10005	10005	3600	10.000
10004	10005	3500	0.000
NULL	NULL	NULL	NULL

Рис. 7.15. Скріншот виконання та результати процедури set_arrival

Перевіримо, що буде, якщо дані не вірні(рис. 7.16):

32 22:10:20 CALL set_arrival(null, 1, 1, 300000, "якийсь опис", 10015, 3600, 10) Error Code: 1644. Помилка, goods_dscr: запчастина з id = 10015 не існує

9 •

SELECT * FROM arrivals ORDER BY arrival_id DESC LIMIT 2;

result Grid

Filter Rows:

Edit:

Export/Import

arrival_id	customer_id	employee_id	date_arrival	total	description
1009	1	1	NULL	300000	якийсь опис
1003	266	1	2024-11-06	260000	NULL
NULL	NULL	NULL	NULL	NULL	NULL

10 •

SELECT *

11 FROM arrivals_goods

12 ORDER BY arrival_id

13 DESC LIMIT 2;

Result Grid

Filter Rows:

arrival_id	goods_id	quantity
1009	10005	10.000
1003	6	20.000
NULL	NULL	NULL

6 •

SELECT * FROM goods

7 ORDER BY goods_id DESC

8 LIMIT 2;

result Grid

Filter Rows:

Edit:

goods_id	goods_dscr_id	price_in	stock	reserve
10005	10005	3600	10.000	0.000
10004	10005	3500	0.000	0.000
NULL	NULL	NULL	NULL	NULL

Рис. 7.16. Скріншот виконання та результати процедури set_arrival при помилкових даних

В таблицях без змін, ROLLBACK відпрацював вірно.

7.3.3. Оформлення замовлення. Лістинг коду процедури set_order та скріншоти виконання (рис. 7.17-7.21):

```
DELIMITER $$
DROP PROCEDURE IF EXISTS set_order $$
CREATE PROCEDURE set_order(
    IN order_id INT,
    IN customer_id INT,
    IN description VARCHAR(255),
    IN brand_id SMALLINT,
    IN rc_employee_id TINYINT,
    IN model_name VARCHAR(20),
    IN sn VARCHAR(30),
    IN equipment VARCHAR(200),
    IN fault_description VARCHAR(200),
    IN rp_employee_id TINYINT,
    IN new_goods_id INT,
    IN quantity decimal(7,3)
)

COMMENT 'Створення нового замовлення, запис у 4 таблиці в одній транзакції'
-- CALL set_order(null, 1, "Глючить на великих тиражах", 3, 3, "P3005", "123WDR2312",
"без кабелю живлення", "тріск", 3, 3, 1);

LANGUAGE SQL
DETERMINISTIC
MODIFIES SQL DATA

BEGIN

    DECLARE new_order_id INT;
    DECLARE order_date DATE;
    DECLARE message VARCHAR(200);

    SET new_order_id = order_id;
    SET order_date = CURDATE();

    START TRANSACTION;

    -- записуємо дані в 4 таблиці
    IF order_id IS NULL THEN
        INSERT INTO orders(customer_id, date_invoice, description)
        VALUE(customer_id, order_date, description);
```

```

SET new_order_id = LAST_INSERT_ID();
END IF;

IF brand_id THEN -- вставляємо, якщо є дані (<> 0) в запиті
    INSERT INTO orders_received(order_id, brand_id, employee_id, model_name,
sn, equipment, fault_description)
    VALUES (new_order_id, brand_id, rc_employee_id, model_name, sn,
equipment, fault_description);
END IF;

IF rp_employee_id THEN -- вставляємо, якщо є дані (<> 0) в запиті
    INSERT INTO orders_repair(order_id, employee_id)
    VALUES (new_order_id, rp_employee_id);
END IF;

IF new_goods_id IS NOT NULL THEN -- вставляємо, якщо є дані в запиті
    IF EXISTS (SELECT 1 FROM goods WHERE goods_id = new_goods_id)
    THEN
        INSERT INTO orders_goods(order_id, goods_id, quantity)
        VALUES (new_order_id, new_goods_id, quantity);
    ELSE
        SET message = CONCAT('Помилка, goods: запчастина з id = ',
new_goods_id, ' не існує');
        ROLLBACK;

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

COMMIT;
SET autocommit = 1;
END $$
DELIMITER ;

```

✓ 39 22:38:02 CALL set_order(null, 1, "Глючить на великих тиражах", 3, 3, "P3005", "123WDR2312", "без кабелю живлення", "тріск", 3, 3, 1) 0 row(s) affected 0.047 sec

16 • SELECT * FROM orders ORDER BY order_id DESC LIMIT 2;

order_id	customer_id	employee_id	delivery_id	date_invoice	date_delivery	delivery_number	date_returned	total	description
10038	1	NULL	NULL	2025-01-02	NULL	NULL	NULL	4137	Глючить на великих тиражах
10037	4	NULL	NULL	2024-12-28	NULL	NULL	NULL	4686	Користувач просить перевіри...

17 • SELECT * FROM orders_received ORDER BY order_id DESC LIMIT 2;

order_id	brand_id	employee_id	model_name	sn	barcode	equipment	fault_description
10038	3	3	P3005	123WDR2312	NULL	без кабелю живлення	тріск
10037	3	4	3015	123--341	NULL	тільки принтер	тріск

18 • `SELECT * FROM orders_repair ORDER BY order_id DESC LIMIT 2;`

order_id	employee_id	date_ready	price_repair	description
10038	3	NULL	0	NULL
10037	4	NULL	0	NULL
NULL	NULL	NULL	NULL	NULL

19 • `SELECT * FROM orders_goods ORDER BY order_id DESC LIMIT 2;`

order_id	goods_id	quantity	price_out	description	is_shipped
10038	3	1.000	4137	NULL	0
10037	4	1.000	4686	NULL	0
NULL	NULL	NULL	NULL	NULL	NULL

Рис. 7.17. Скріншот виконання та результати процедури set_order

✖ 46 22:54:03 CALL set_order(null, 5, "Стукає", 3, 5, "P3015", "123WDR23TT", "без кабелю живлення", "стук", 5, 5, 1) Error Code: 1048. Column 'price_out' cannot be null
 ✖ 49 22:57:11 CALL set_order(null, 5, "Стукає", 3, 5, "P3015", "123WDR23TT", "без кабелю живлення", "стук", 5, 5, 1) Error Code: 1644. Помилка, goods: запчастина з id = 5 не існує

16 • `SELECT * FROM orders ORDER BY order_id DESC LIMIT 2;`

order_id	customer_id	employee_id	delivery_id	date_invoice	date_delivery	delivery_number	date_returned	total	description
10039	5	NULL	NULL	2025-01-02	NULL	NULL	NULL	NULL	Стукає
10038	1	NULL	NULL	2025-01-02	NULL	NULL	NULL	4137	Глючить на великих тиражах
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

17 • `SELECT * FROM orders_received ORDER BY order_id DESC LIMIT 2;`

order_id	brand_id	employee_id	model_name	sn	barcode	equipment	fault_description
10039	3	5	P3015	123WDR23TT	NULL	без кабелю живлення	стук
10038	3	3	P3005	123WDR2312	NULL	без кабелю живлення	тріск
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

18 • `SELECT * FROM orders_repair ORDER BY order_id DESC LIMIT 2;`

order_id	employee_id	date_ready	price_repair	description
10039	5	NULL	0	NULL
10038	3	NULL	0	NULL
NULL	NULL	NULL	NULL	NULL





Result Grid						
Filter Rows: <input type="text"/>						
Edit:   						
Export/Import: 						
order_id	goods_id	quantity	price_out	descriptio	is_shipped	
10038	3	1.000	4137	NULL	0	
10037	4	1.000	4686	NULL	0	
NULL	NULL	NULL	NULL	NULL	NULL	

Рис. 7.18. Скріншот виконання та результати процедури `set_order` при помилкових даних

Бачимо, що не дивлячись на помилку при вставці в останню таблицю, в перші три таблиці дані були додані (замовлення з `id = 10039`).

Додам в код процедури SET autocommit = 0;(рис. 7.19):

```
BEGIN

DECLARE new_order_id INT;
DECLARE order_date DATE;
DECLARE message VARCHAR(200);

SET new_order_id = order_id;
SET order_date = CURDATE();
SET autocommit = 0;

START TRANSACTION;
```

Рис. 7.19. Скріншот кода процедури

Повторимо експеримент(рис. 7.20)

61 23:24:13 CALL set_order(null, 5, "Провести ТО", 3, 5, "Р3015", "123WDR23TT", "без кабелю живлення", "ТО", 50, 5, 1) Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails ('ss_innodb`.`orders` repair...

[illegible][illegible]

18 • `SELECT * FROM orders_repair ORDER BY order_id DESC LIMIT 2;`

order_id	employee_id	date_ready	price_repair	description
10039	5	NULL	0	NULL
10038	3	NULL	0	NULL
NULL	NULL	NULL	NULL	NULL

Рис. 7.20. Скріншот виконання та результати процедури set_order при помилкових даних

Бачимо, що помилка при вставці таблиці не відкотила усю транзакцію, але тільки частину, все що було до помилки, додалось. Щоб цього уникнути треба робити перевірку (верифікацію) даних, що додаються, якщо є помилки, робити ROLLBACK. Тоді транзакція буде мати вірний відкат.

Додамо до процедури такі перевірки, та виконаємо її знов (рис.7.21)

77 23:54:19 CALL set_order(null, 6, "ТО", 3, 6, "2055", "1QWER23TT", "без кабелю живлення", "поганий картридж", 50, 6, 1) Error Code: 1644. Помилка, employees: робітника з id = 50 не існує

16 • `SELECT * FROM orders ORDER BY order_id DESC LIMIT 2;`

order_id	customer_id	employee_id	delivery_id	date_invoice	date_delivery	delivery_number	date_returned	total	description
10042	5	NULL	NULL	2025-01-02	NULL	NULL	NULL	NULL	Провести ТО
10039	5	NULL	NULL	2025-01-02	NULL	NULL	NULL	NULL	Стукає
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

17 • `SELECT * FROM orders_received ORDER BY order_id DESC LIMIT 2;`

order_id	brand_id	employee_id	model_name	sn	barcode	equipment	fault_descripti
10042	3	5	P3015	123WDR23TT	NULL	без кабелю живлення	ТО
10039	3	5	P3015	123WDR23TT	NULL	без кабелю живлення	стук
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

18 • `SELECT * FROM orders_repair ORDER BY order_id DESC LIMIT 2;`

order_id	employee_id	date_ready	price_repair	description
10039	5	NULL	0	NULL
10038	3	NULL	0	NULL
NULL	NULL	NULL	NULL	NULL

19 • `SELECT * FROM orders_goods ORDER BY order_id DESC LIMIT 2;`

order_id	goods_id	quantity	price_out	descriptio	is_shipped
10038	3	1.000	4137	NULL	0
10037	4	1.000	4686	NULL	0
NULL	NULL	NULL	NULL	NULL	NULL

Рис. 7.21. Скріншот виконання та результати процедури set_order при помилкових даних

Тепер помилку було оброблено вірно, нових записів до таблиць додано не було. Виправлений код процедури:

```
DELIMITER $$
DROP PROCEDURE IF EXISTS set_order $$
CREATE PROCEDURE set_order(
    IN order_id INT,
    IN new_customer_id INT,
    IN description VARCHAR(255),
    IN new_brand_id SMALLINT,
    IN rc_employee_id TINYINT,
    IN model_name VARCHAR(20),
    IN sn VARCHAR(30),
    IN equipment VARCHAR(200),
    IN fault_description VARCHAR(200),
    IN rp_employee_id TINYINT,
    IN new_goods_id INT,
    IN quantity decimal(7,3)
)
COMMENT ' Створення нового замовлення, запис у 4 таблиці в одній транзакції '
-- CALL set_order(null, 1, "Глючить на великих тиражах", 3, 3, "P3005" , "123WDR2312", "без кабелю живлення", "тріск", 3, 3, 1);

LANGUAGE SQL
DETERMINISTIC
MODIFIES SQL DATA

BEGIN

    DECLARE new_order_id INT;
    DECLARE order_date DATE;
    DECLARE message VARCHAR(200);

    SET new_order_id = order_id;
    SET order_date = CURDATE();

    START TRANSACTION;

    -- записуємо дані в 4 таблиці
```

```

IF order_id IS NULL THEN
    IF EXISTS (SELECT 1 FROM customers WHERE customer_id =
new_customer_id) THEN
        INSERT INTO orders(customer_id, date_invoice, description)
        VALUE(new_customer_id, order_date, description);
        SET new_order_id = LAST_INSERT_ID();
    ELSE
        SET message = CONCAT('Помилка, customers: клієнта з id = ',
new_customer_id, ' не існує');
        ROLLBACK;

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

IF new_brand_id IS NOT NULL THEN -- вставляємо, якщо є дані в запиті
    IF EXISTS (SELECT 1 FROM brand WHERE brand_id = new_brand_id) THEN
        IF EXISTS (SELECT 1 FROM employees WHERE employee_id =
rc_employee_id) THEN
            INSERT INTO orders_received(order_id, brand_id, employee_id,
model_name, sn, equipment, fault_description)
            VALUES (new_order_id, new_brand_id, rc_employee_id,
model_name, sn, equipment, fault_description);
        ELSE
            SET message = CONCAT('Помилка, employees: робітника з id = ',
rc_employee_id, ' не існує');
            ROLLBACK;

            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = message;
        END IF;
    ELSE
        SET message = CONCAT('Помилка, brand: brand з id = ',
new_brand_id, ' не існує');
        ROLLBACK;

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

IF rp_employee_id IS NOT NULL THEN -- вставляємо, якщо є дані в запиті
    IF EXISTS (SELECT 1 FROM employees WHERE employee_id =
rp_employee_id) THEN
        INSERT INTO orders_repair(order_id, employee_id)
        VALUES (new_order_id, rp_employee_id);
    ELSE
        SET message = CONCAT('Помилка, employees: робітника з id = ',
rp_employee_id, ' не існує');
        ROLLBACK;

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

```

```

IF new_goods_id IS NOT NULL THEN -- вставляємо, якщо є дані в запиті
    IF EXISTS (SELECT 1 FROM goods WHERE goods_id = new_goods_id)
    THEN
        INSERT INTO orders_goods(order_id, goods_id, quantity)
        VALUES (new_order_id, new_goods_id, quantity);
    ELSE
        SET message = CONCAT('Помилка, goods: запчастина з id = ',
new_goods_id, ' не існує');
        ROLLBACK;

        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = message;

    END IF;
END IF;

COMMIT;

END $$
DELIMITER ;

```

Але, мене бентежить один момент, коли помилка прийде ззовні, наприклад, із тригера. Проведемо ще один експеримент, додамо в код процедури такий код:

```

DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
BEGIN
    ROLLBACK;
    SET message = 'Транзакцію скасовано через помилку';
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = message;
END;

```

Виконаємо процедуру(рис.7.22):

```
CALL set_order(null, 6, "TO", 30, 6, "2055" , "1QWER23TT", "без кабелю живлення",
"поганий картридж", 50, 6, 1);
```

До

16 • SELECT * FROM orders ORDER BY order_id DESC LIMIT 2;									
Result Grid Filter Rows: Edit: Export/Import: Wrap Cell Content:									
order_id	customer_id	employee_id	delivery_id	date_invoice	date_delivery	delivery_number	date_returned	total	description
10042	5	NULL	NULL	2025-01-02	NULL	NULL	NULL	NULL	Провести ТО
10039	5	NULL	NULL	2025-01-02	NULL	NULL	NULL	NULL	Стукає
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
✖	22	14:41:16	CALL set_order(null, 6, "TO", 30, 6, "2055", "1QWER23TT", "без кабелю живлення", "поганий картридж", 5, 5, 1)					Error Code: 1644. Помилка brand: brand s id = 30 не існує	
✖	23	14:41:29	CALL set_order(null, 6, "TO", 3, 6, "2055", "1QWER23TT", "без кабелю живлення", "поганий картридж", 5, 5, 1)					Error Code: 1644. Помилка goods: запчастина з id = 5 не існує	
✖	25	14:41:53	CALL set_order(null, 6, "TO", 3, 6, "2055", "1QWER23TT", "без кабелю живлення", "поганий картридж", 5, 7, 100)					Error Code: 1644. Транзакцію скасовано через помилку	

Після

16 • `SELECT * FROM orders ORDER BY order_id DESC LIMIT 2;`

order_id	customer_id	employee_id	delivery_id	date_invoice	date_delivery	delivery_number	date_returned	total	description
10042	5	NULL	NULL	2025-01-02	NULL	NULL	NULL	NULL	Провести ТО
10039	5	NULL	NULL	2025-01-02	NULL	NULL	NULL	NULL	Стукає
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

17 • `SELECT * FROM orders_received ORDER BY order_id DESC LIMIT 2;`

order_id	brand_id	employee_id	model_name	sn	barcode	equipment	fault_description
10042	3	5	P3015	123WDR23TT	NULL	без кабелю живлення	ТО
10039	3	5	P3015	123WDR23TT	NULL	без кабелю живлення	стук
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Рис. 7.22 Скріншот виконання процедури з універсальним обробником помилок (остання помилка прийшла ззовні, з триггеру).

Ну що ж, знайдено універсальний обробник помилок, який гарантовано веде до ROLLBACK при появі помилки! Чудово!

7.3.4. Бінарний журнал СУБД. MySQL. Виконаємо код, а потім переглянемо файл журналу за допомогою Workbench:

```
"c:\Program Files\MySQL\MySQL Server 8.0\bin\mysqlbinlog.exe"  
"c:\ProgramData\MySQL\MySQL Server 8.0\Data\IHOR-bin.000151" > 151.sql
```

Назва файлу журналу у мене має вигляд «IHOR-bin.000151». Файл створюється на початку сесії, та завершується наприкінці сесії. Термін зберігання файлів – 1 місяць. Це можна змінити.

Альтернативний виклик журналу виконавши код у терміналі з папки «Data»(рис. 7.23):

```
"c:\Program Files\MySQL\MySQL Server 8.0\bin\mysqlbinlog.exe" IHOR-bin.000151
```



```

user@Ihor MINGW64 /c/ProgramData/MySQL/MySQL Server 8.0/Data
$ "c:\Program Files\MySQL\MySQL Server 8.0\bin\mysqlbinlog.exe" IHOR-bin.000151
# The proper term is pseudo_replica_mode, but we use this compatibility alias
# to make the statement usable on server versions 8.0.24 and older.
/*!50530 SET @@SESSION.PSEUDO_SLAVE_MODE=1*/;
/*!50003 SET @OLD_COMPLETION_TYPE=@@COMPLETION_TYPE,COMPLETION_TYPE=0*/;
DELIMITER /*!*/;
# at 4
#250102 13:45:24 server id 1  end_log_pos 126 CRC32 0x39e7d214  Start: binlog v
4, server v 8.0.33 created 250102 13:45:24 at startup
# Warning: this binlog is either in use or was not closed properly.
ROLLBACK/*!*/;
BINLOG '
VHx2Zw8BAAAAegAAAH4AAAAABAAQAOC4wLjMzAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAABUfHZnEwANAAgAAAAABAAEAAAYgAEGggAAAAICAgCAAAACgoKKioAEjQA
CigAARTS5zk=
'/*!*/;
# at 126
#250102 13:45:24 server id 1  end_log_pos 157 CRC32 0xc71ba0aa  Previous-GTIDs
# [empty]
# at 157
#250102 15:34:13 server id 1  end_log_pos 236 CRC32 0x753ee211  Anonymous_GTID 1
ast_committed=0 sequence_number=1      rbr_only=no      original_committed_times
tamp=1735824853271546  immediate_commit_timestamp=1735824853271546  transact
ion_length=1540
# original_commit_timestamp=1735824853271546 (2025-01-02 15:34:13.271546 q
(
))
# immediate_commit_timestamp=1735824853271546 (2025-01-02 15:34:13.271546 q
(
))
/*!80001 SET @@session.original_commit_timestamp=1735824853271546/*!*/;
/*!80014 SET @@session.original_server_version=80033/*!*/;
/*!80014 SET @@session.immediate_server_version=80033/*!*/;
SET @@SESSION.GTID_NEXT= 'ANONYMOUS'/*!*/;
# at 236
#250102 15:34:13 server id 1  end_log_pos 1697 CRC32 0xcc0f3033      Query t
hread_id=9      exec_time=0      error_code=0      Xid = 94
use `ss_innodb`/*!*/;
SET TIMESTAMP=1735824853/*!*/;
SET @@session.pseudo_thread_id=9/*!*/;
SET @@session.foreign_key_checks=1, @@session.sql_auto_is_null=0, @@session.unio
ue_checks=1, @@session.autocommit=1/*!*/;
SET @@session.sql_mode=1168113696/*!*/;
SET @@session.auto_increment_increment=1, @@session.auto_increment_offset=1/*!*/
;
/*!\C utf8mb4 *//*!*/;
SET @@session.character_set_client=255,@@session.collation_connection=255,@@sess
ion.collation_server=255/*!*/;
SET @@session.lc_time_names=0/*!*/;
SET @@session.collation_database=DEFAULT/*!*/;
/*!80011 SET @@session.default_collation_for_utf8mb4=255/*!*/;
CREATE DEFINER='root'@'localhost' PROCEDURE `set_goods_dscr`(
    IN unit_id tinyint,
    IN group_id smallint,
    IN part_number CHAR(10),
    IN name VARCHAR(50),
    IN margin tinyint,
    IN new_price int,
    IN description VARCHAR(255),
    IN price_in INT
)
MODIFIES SQL DATA
DETERMINISTIC
COMMENT 'Створення нового найменування товару, запис у 2 таблиці в одной тра

```

Рис. 7.23 Скріншот виклику та вмісту бінарного журналу.

Наприклад, подивимось на такий фрагмент коду:

```
# at 12923 #250103 14:40:49 server id 1 end_log_pos 13000 CRC32 0x2ab5fb06
Anonymous_GTID last_committed=6 sequence_number=7 rbr_only=no
original_committed_timestamp=1735908049717705
immediate_commit_timestamp=1735908049717705 transaction_length=207 #
original_commit_timestamp=1735908049717705 (2025-01-03 14:40:49.717705 Z) #
immediate_commit_timestamp=1735908049717705 (2025-01-03 14:40:49.717705 Z) #
/*!80001 SET @@session.original_commit_timestamp=1735908049717705/*!*/; /*!80014
SET @@session.original_server_version=80033/*!*/; /*!80014 SET
@@session.immediate_server_version=80033/*!*/; SET @@SESSION.GTID_NEXT=
'ANONYMOUS'/*!*/;
```

- **# at 12923** – Це позиція у файлі бінарного журналу, з якої починається цей запис.
- **#250103 14:40:49** – Дата і час виконання транзакції. У цьому випадку це 3 січня 2025 року о 14:40:49.
- **server id 1** – Ідентифікатор сервера, на якому виконувалася транзакція. Це важливо в реплікації, щоб відрізнити транзакції з різних серверів.
- **end_log_pos 13000** – Позиція, на якій завершується цей запис у binlog.
- **CRC32 0x2ab5fb06** – Контрольна сума (CRC32) для перевірки цілісності запису.
- **Anonymous_GTID** – Вказує, що транзакція має анонімний GTID (Global Transaction Identifier). Це тип GTID, що не прив'язаний до конкретного сервера, часто використовується, коли GTID не увімкнено.
- **last_committed=6 sequence_number=7** –
 - **last_committed=6** – Ідентифікатор останньої підтвердженої транзакції.
 - **sequence_number=7** – Порядковий номер цієї транзакції.
- **rbr_only=no** – Вказує, що це не лише транзакція на рівні рядків (Row-Based Replication), можливо, змішаного типу або Statement-Based.
- **original_committed_timestamp=1735908049717705** – Час виконання транзакції у мікросекундах з епохи Unix (тобто, 14:40:49.717705).
- **immediate_commit_timestamp=1735908049717705** – Час негайного коміту транзакції, збігається з **original_committed_timestamp**.

Команди налаштування сесії:

1. **/*!80001 SET @@session.original_commit_timestamp=1735908049717705/*!*/;**
 - Встановлює timestamp транзакції для сесії. 80001 означає, що ця команда буде виконана тільки на MySQL 8.0.1 і вище.
2. **/*!80014 SET @@session.original_server_version=80033/*!*/;**

- Встановлює версію сервера (8.0.33).
- 3. `/*!80014 SET @@session.immediate_server_version=80033*/;`
 - Встановлює версію сервера для негайного коміту транзакції.
- 4. `SET @@SESSION.GTID_NEXT= 'ANONYMOUS'`
 - Вказує, що наступна транзакція буде з анонімним GTID.

Як знайти зміст транзакції:

1. **Перегляд binlog:** Щоб побачити, які саме SQL-запити виконувала ця транзакція, потрібно розшифрувати наступні записи binlog після цієї мета-інформації. Для цього використовується команда:

```
mysqlbinlog --base64-output=DECODE-ROWS -v /path/to/binlog.000001
```

Опції:

- `--base64-output=DECODE-ROWS` – декодує записи на рівні рядків (RBR).
- `-v` – показує SQL-запити в читабельному вигляді.
- `-vv` – ще детальніший вивід.

Спробуємо, для цього виконаємо команди:

`CALL set_order(10038, 6, "TO", null, 6, "2055" , "1QWER23TT", "без кабелю живлення", "поганий картридж", null, 7, 1);`

Цей код додав запчастину з `id = 7` до таблиці `orders_goods` (`id=10038`), та тригер додав в резерв (таблиця `goods`) 1 шт цієї запчастини, а також тригери змінили поля `price_out` (таблиця `orders_goods`) та `total` (таблиця `orders`).

`"c:\Program Files\MySQL\MySQL Server 8.0\bin\mysqlbinlog.exe" --base64-output=DECODE-ROWS -v "C:\ProgramData\MySQL\MySQL Server 8.0\Data\IHOR-bin.000152" > 152.sql`

19 • `SELECT * FROM orders_goods ORDER BY order_id DESC LIMIT 3;`

order_id	goods_id	quantity	price_out	descriptio	is_shipped
10038	7	1.000	6041	NULL	0
10038	3	1.000	4137	NULL	0
10037	4	1.000	4686	NULL	0
NULL	NULL	NULL	NULL	NULL	NULL

21 • `SELECT * FROM goods WHERE goods_id = 7;`

goods_id	goods_dscr_id	price_in	stock	reserve
7	7	4646	3.000	1.000
NULL	NULL	NULL	NULL	NULL

Подивимось на останню транзакцію в бінарному журналі:

```
# at 16802
#250103 15:26:29 server id 1 end_log_pos 16881 CRC32 0xdb3ed632      Anonymous_GTID
      last_committed=8          sequence_number=9          rbr_only=yes
      original_committed_timestamp=1735910789446891 immediate_commit_timestamp=1735910789446891
      transaction_length=731
/*!50718 SET TRANSACTION ISOLATION LEVEL READ COMMITTED*//*!*/;
# original_commit_timestamp=1735910789446891 (2025-01-03 15:26:29.446891 ㄗㄘㄘ (話→
# immediate_commit_timestamp=1735910789446891 (2025-01-03 15:26:29.446891 ㄗㄘㄘ (話→
/*!80001 SET @@session.original_commit_timestamp=1735910789446891*//*!*/;
/*!80014 SET @@session.original_server_version=80033*//*!*/;
/*!80014 SET @@session.immediate_server_version=80033*//*!*/;
SET @@SESSION.GTID_NEXT= 'ANONYMOUS'/*!*/;
# at 16881
#250103 15:26:29 server id 1 end_log_pos 16980 CRC32 0x6a3c9683      Query  thread_id=9
      exec_time=0      error_code=0
SET TIMESTAMP=1735910789/*!*/;
SET @@session.time_zone='SYSTEM'/*!*/;
BEGIN
/*!*/;
# at 16980
#250103 15:26:29 server id 1 end_log_pos 17055 CRC32 0xf75b5d52      Table_map:
`ss_innodb`.`orders_goods` mapped to number 110
# has_generated_invisible_primary_key=0
# at 17055
#250103 15:26:29 server id 1 end_log_pos 17119 CRC32 0xe67c02a1      Table_map: `ss_innodb`.`goods`
mapped to number 114
# has_generated_invisible_primary_key=0
# at 17119
#250103 15:26:29 server id 1 end_log_pos 17195 CRC32 0xb2970b84      Table_map: `ss_innodb`.`orders`
mapped to number 99
# has_generated_invisible_primary_key=0
# at 17195
#250103 15:26:29 server id 1 end_log_pos 17273 CRC32 0x37723e41      Update_rows: table id 114
# at 17273
#250103 15:26:29 server id 1 end_log_pos 17326 CRC32 0xdfdc1a76      Write_rows: table id 110
# at 17326
#250103 15:26:29 server id 1 end_log_pos 17502 CRC32 0xdaa47b25      Update_rows: table id 99 flags:
STMT_END_F
### UPDATE `ss_innodb`.`goods`
### WHERE
###   @1=7
###   @2=7
###   @3=4646
###   @4=3.000
###   @5=0.000
### SET
###   @1=7
###   @2=7
###   @3=4646
```

```

### @4=3.000
### @5=1.000
### INSERT INTO `ss_innodb`.`orders_goods`
### SET
### @1=10038
### @2=7
### @3=1.000
### @4=6041
### @5=NULL
### @6=0
### UPDATE `ss_innodb`.`orders`
### WHERE
### @1=10038
### @2=1
### @3=NULL
### @4=NULL
### @5='2025:01:02'
### @6=NULL
### @7=NULL
### @8=NULL
### @9=4137
### @10='Глючить на великих тиражах'
### @11=0
### @12=NULL
### SET
### @1=10038
### @2=1
### @3=NULL
### @4=NULL
### @5='2025:01:02'
### @6=NULL
### @7=NULL
### @8=NULL
### @9=10178
### @10='Глючить на великих тиражах'
### @11=0
### @12=NULL
# at 17502
#250103 15:26:29 server id 1 end_log_pos 17533 CRC32 0xdc03bc8e Xid = 430
COMMIT/*!*/;
SET @@SESSION.GTID_NEXT= 'AUTOMATIC' /* added by mysqlbinlog */ /*!*/;
DELIMITER ;
# End of log file
/*!50003 SET COMPLETION_TYPE=@OLD_COMPLETION_TYPE*/;
/*!50530 SET @@SESSION.PSEUDO_SLAVE_MODE=0*/;

```

Що бачимо в журналі:

- о 2025-01-03 15:26:29 відбулася транзакція з рівнем ізоляції **READ COMMITTED**;
- внесені зміни в три таблиці:
 - goods, додав резерв (@5=1.000) до goods_id = 7;
 - orders_goods, додав рядок з goods_id = 7 та order_id = 10038 та ціною @4=6041;
 - orders, була зміна в рядку (тригер змінив поле total, після вставки рядка в orders_goods), о цьому говорить прапорець STMT_END_F;
- COMMIT, зміни підтверджені.

Висновки.

- Розібралися з блокуванням таблиць з движком MyISAM, переконалися, що з іншої сесії доступ до БД обмежено в залежності от типа блокування, або є доступ на читання, або доступу немає взагалі. Це блокування тимчасове, як тільки інший сеанс знімає блокування, одразу розблокований сеанс отримує доступ до БД і може виконати свої запити до БД.
- Розробили три транзакції для БД InnoDB, які здійснюють додавання даних одразу в кілька таблиць БД.
- Перевірили роботу кожної транзакції при коректних та помилкових даних, побачили як працює ROLLBACK.
- Провели кілька експериментів з помилковими даними, з помилками, що виникають, як всередині процедури, так і ззовні (наприклад, в тригері). Оцінили вплив помилок на ROLLBACK. Додали універсальний обробник помилок, який коректно перериває транзакцію.
- Розглянули бінарний журнал транзакцій, навчилися працювати з ним і отримувати інформацію, які були транзакції, що змінювали в БД.