BNCL / COMP2350 Assignment 2 Report

Unit Code	**COMP2350 & COMP6350**	**Assignment#**	**2**
Student ID Number	**48476595**	**Student Name**	**Tuan Son Hoang**
Student ID Number	**48724831**	**Student Name**	**Niraj Arun Bhoi**
Student ID Number	**47834250**	**Student Name**	**Bulgantamir Nyamdeleg**
Student ID Number	**48795240**	**Student Name**	**Kirtan Manoj Suthar**
Tutor's Name	**2**	**Workshop Date/Time**	**Workshop_1-29**

Section 1: Task 2.3 – Function Testing

Functions covered

- `calcLoyaltyPoints(orderID)`: earn *1 point per $1.*

  Awarded *only when* `CusOrder.orderStatus='Delivered'`; result is *floored to an integer.* Returns `0` for non-`Delivered` or missing orders.

- `isGiftCardValid(code)`: returns `1` if a gift card with the code *exists* and `isActive=1` and `expirationDate >= CURDATE()`; otherwise `0`. (Balance is not part of validity.)

Scenarios we tested

A) `calcLoyaltyPoints(orderID)` (using a dedicated test user)

- `Delivered`, total `79.40` → `79` pts (floor).
- `Processing`, total `79.40` → `0` pts.
- `Cancelled`, total `120.00` → `0` pts.
- `Returned`, total `45.50` → `0` pts.
- `Delivered`, total `0.99` → `0` pts (floor from $0.99).
- `Delivered`, total `100.00` → `100` pts.
- Non-existent order ID → `0` pts.

B) `isGiftCardValid(code)` (cards created for the same test user)

- `S_TODAY`: active, expires today → `1`.
- `S_FUTURE`: active, future expiry → `1`.
- `S_YDAY`: active, *expired yesterday* → `0`.
- `S_OFF`: *inactive* but not expired → `0`.
- `S_ZERO`: active, future expiry, *zero balance* → `1` (still valid).
- `NOPE`: code not found → `0`.
- empty string `''` → `0`.

Test SQL:

```
-- Make a test user (ignore if already exists)
INSERT IGNORE INTO `User` (userName, email, userPassword, phone,
   loyaltyPoints, isMember)
VALUES ('T23_Simple', 't23_simple@example.com', 'x', '000', 100, 1);

-- ------------------------------------------------
-- A) calcLoyaltyPoints(orderID)
-- Assumptions: earn 1 point per $1, award only when Delivered, floor to
   integer.

-- Build orders with known totals/status for this user
INSERT INTO CusOrder (userID, totalAmount, orderStatus)
VALUES
((SELECT userID FROM `User` WHERE email='t23_simple@example.com' LIMIT 1),
    79.40,  'Delivered'),
((SELECT userID FROM `User` WHERE email='t23_simple@example.com' LIMIT 1),
    79.40,  'Processing'),
((SELECT userID FROM `User` WHERE email='t23_simple@example.com' LIMIT 1),
    120.00, 'Cancelled'),
((SELECT userID FROM `User` WHERE email='t23_simple@example.com' LIMIT 1),
    45.50,  'Returned'),
((SELECT userID FROM `User` WHERE email='t23_simple@example.com' LIMIT 1),
    0.99,   'Delivered'),
((SELECT userID FROM `User` WHERE email='t23_simple@example.com' LIMIT 1),
    100.00, 'Delivered');

-- Show results for ALL the above orders (easy screenshot)
SELECT orderID, orderStatus, totalAmount,
       calcLoyaltyPoints(orderID) AS pts
```

```
FROM CusOrder
WHERE userID = (SELECT userID FROM `User` WHERE email='t23_simple@example.com' LIMIT 1)
ORDER BY orderID;

-- Non-existent order test (should be 0)
SELECT 'Nonexistent -> 0' AS case_desc, calcLoyaltyPoints(99999999) AS pts;

-- ------------------------------------------
-- B) isGiftCardValid(code)
-- Valid if: card exists AND isActive=1 AND expirationDate >= CURDATE().

-- Create some gift cards for the same user (ignore if they already exist)
INSERT IGNORE INTO GiftCard (giftCardCode, userID, balance, isActive, expirationDate) VALUES
('S_TODAY',  (SELECT userID FROM `User` WHERE email='t23_simple@example.com' LIMIT 1), 25.00, 1, CURDATE()),
('S_FUTURE', (SELECT userID FROM `User` WHERE email='t23_simple@example.com' LIMIT 1), 50.00, 1, DATE_ADD(CURDATE(), INTERVAL 30 DAY)),
('S_YDAY',   (SELECT userID FROM `User` WHERE email='t23_simple@example.com' LIMIT 1), 50.00, 1, DATE_SUB(CURDATE(), INTERVAL 1 DAY)),
('S_OFF',    (SELECT userID FROM `User` WHERE email='t23_simple@example.com' LIMIT 1), 50.00, 0, DATE_ADD(CURDATE(), INTERVAL 30 DAY)),
('S_ZERO',   (SELECT userID FROM `User` WHERE email='t23_simple@example.com' LIMIT 1),  0.00, 1, DATE_ADD(CURDATE(), INTERVAL 30 DAY));

-- One-line checks
SELECT
  t.case_desc,
  t.code,
  t.expected_valid,
  isGiftCardValid(t.code) AS actual_valid,
  CASE
    WHEN isGiftCardValid(t.code) = t.expected_valid THEN 'PASS'
    ELSE 'FAIL'
  END AS result
FROM (
  SELECT 1 AS idx, 'TODAY -> 1'     AS case_desc, 'S_TODAY'  AS code, 1 AS expected_valid
  UNION ALL
  SELECT 2, 'FUTURE -> 1',          'S_FUTURE',               1
  UNION ALL
  SELECT 3, 'YESTERDAY -> 0',       'S_YDAY',                 0
```

```
  UNION ALL
  SELECT 4, 'INACTIVE -> 0',          'S_OFF',                  0
  UNION ALL
  SELECT 5, 'ZERO -> 1',              'S_ZERO',                 1
  UNION ALL
  SELECT 6, '404 -> 0',               'NOPE',                   0
  UNION ALL
  SELECT 7, 'EMPTY -> 0',             '',                       0
) AS t
ORDER BY t.idx;
```

orderID	orderStatus	totalAmount	pts
36	Delivered	79.40	79
37	Processing	79.40	0
38	Cancelled	120.00	0
39	Returned	45.50	0
40	Delivered	0.99	0
41	Delivered	100.00	100
43	Delivered	79.40	79
44	Processing	79.40	0
45	Cancelled	120.00	0
46	Returned	45.50	0
47	Delivered	0.99	0
48	Delivered	100.00	100
49	Delivered	79.40	79
50	Processing	79.40	0
51	Cancelled	120.00	0
52	Returned	45.50	0
53	Delivered	0.99	0
54	Delivered	100.00	100
55	Delivered	79.40	79
56	Processing	79.40	0
57	Cancelled	120.00	0
58	Returned	45.50	0
59	Delivered	0.99	0
60	Delivered	100.00	100
62	Delivered	79.40	79
63	Processing	79.40	0
64	Cancelled	120.00	0
65	Returned	45.50	0
66	Delivered	0.99	0
67	Delivered	100.00	100
68	Delivered	79.40	79
69	Processing	79.40	0
70	Cancelled	120.00	0
71	Returned	45.50	0
72	Delivered	0.99	0
73	Delivered	100.00	100

Output for calcLoyaltyPoints

case_desc	code	expected_valid	actual_valid	result
TODAY -> 1	S_TODAY	1	1	PASS
FUTURE -> 1	S_FUTURE	1	1	PASS
YESTERDAY -> 0	S_YDAY	0	0	PASS
INACTIVE -> 0	S_OFF	0	0	PASS
ZERO -> 1	S_ZERO	1	1	PASS
404 -> 0	NOPE	0	0	PASS
EMPTY -> 0		0	0	PASS

Output for isGiftCardValid

Figure 1: Task 2.3 Output

Section 2: Task 3.1 – Procedure Design (`redeemGiftCard`)

Design summary:

- **Purpose/BRs**: Deduct from a valid, active, non-expired gift card with sufficient balance and record a `Completed` gift-card payment (supports BR2/BR3/BR4).
- **Inputs**: `p_orderID INT`, `p_giftCardCode VARCHAR`, `p_amount DECIMAL(10,2)`.
- **Outputs**: success/failure..
- **Preconditions**: Order exists; card exists; `isActive=1`; not expired; `balance >= p_amount`.
- **Postconditions**: `GiftCard.balance` reduced by `p_amount`; a `Payment` row is inserted with method "Gift Card" and status `Completed`. Completion of orders still goes via `CheckoutOrder` (enforces BR2/BR5).
- **Failure handling**: `SIGNAL` with messages such as "Gift card not found", "Gift card inactive/-expired", "Insufficient gift card balance", "Overpayment not allowed".

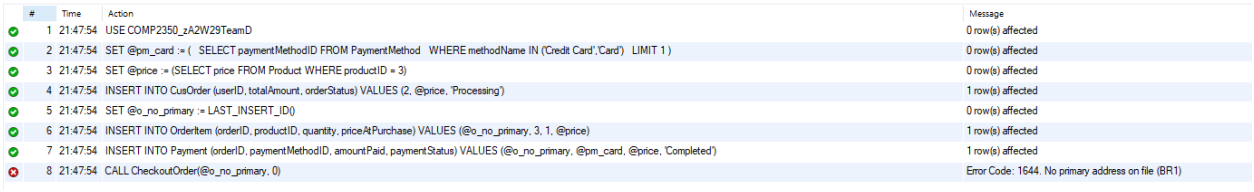
Section 3: Task 3.3 – Procedure Testing (`CheckoutOrder`)

We verify four scenarios: BR1 (no primary address), BR5 (insufficient stock), BR2 (payments $\neq$ total), and a success case.

BR1 – No Primary Address

```
-- pick a payment method id (uses a seeded name)
SET @pm_card := (
  SELECT paymentMethodID FROM PaymentMethod
  WHERE methodName IN ('Credit Card','Card')
  LIMIT 1
);

-- pick a product and remember its price (Cotton T-Shirt, productID = 3)
SET @price := (SELECT price FROM Product WHERE productID = 3);

-- make an order whose total equals the item price
INSERT INTO CusOrder (userID, totalAmount, orderStatus)
VALUES (2, @price, 'Processing');
SET @o_no_primary := LAST_INSERT_ID();

-- add 1 line item with the same price we set above
INSERT INTO OrderItem (orderID, productID, quantity, priceAtPurchase)
VALUES (@o_no_primary, 3, 1, @price);

-- pay exactly the order total (Completed)
INSERT INTO Payment (orderID, paymentMethodID, amountPaid, paymentStatus)
VALUES (@o_no_primary, @pm_card, @price, 'Completed');

```

```
-- expect: Error Code 1644 + message 'No primary address on file (BR1)'
CALL CheckoutOrder(@o_no_primary, 0);
```



#	Time	Action	Message
1	21:47:54	USE COMP2350_zA2W29TeamD	0 row(s) affected
2	21:47:54	SET @pm_card := (SELECT paymentMethodID FROM PaymentMethod WHERE methodName IN ('Credit Card','Card') LIMIT 1)	0 row(s) affected
3	21:47:54	SET @price := (SELECT price FROM Product WHERE productID = 3)	0 row(s) affected
4	21:47:54	INSERT INTO CusOrder (userID, totalAmount, orderStatus) VALUES (2, @price, 'Processing')	1 row(s) affected
5	21:47:54	SET @o_no_primary := LAST_INSERT_ID()	0 row(s) affected
6	21:47:54	INSERT INTO OrderItem (orderID, productID, quantity, priceAtPurchase) VALUES (@o_no_primary, 3, 1, @price)	1 row(s) affected
7	21:47:54	INSERT INTO Payment (orderID, paymentMethodID, amountPaid, paymentStatus) VALUES (@o_no_primary, @pm_card, @price, 'Completed')	1 row(s) affected
8	21:47:54	CALL CheckoutOrder(@o_no_primary, 0)	Error Code: 1644. No primary address on file (BR1)

Figure 2: BR1 Output.

BR5 – Insufficient Stock

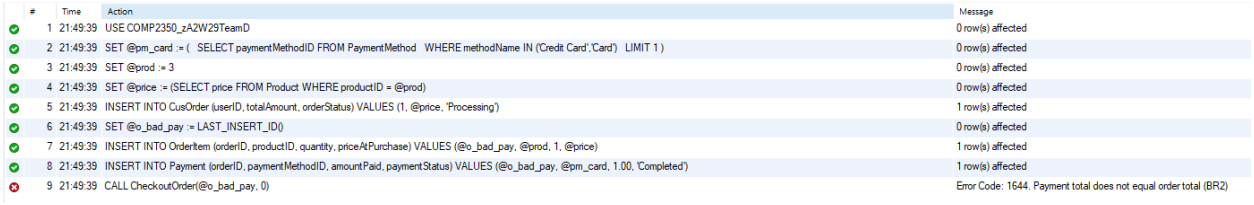
```
-- pick a payment method
SET @pm_card := (
  SELECT paymentMethodID FROM PaymentMethod
  WHERE methodName IN ('Credit Card','Card')
  LIMIT 1
);

-- choose a product and read its price + current stock
-- (productID 11 or 3 works; We'll use 11 as example)
SET @prod := 11;
SET @price := (SELECT price FROM Product WHERE productID = @prod);
SET @stock := (SELECT stockQuantity FROM Product WHERE productID = @prod);

-- make quantity one more than stock to guarantee failure
SET @qty := @stock + 1;

-- user with a primary address (userID = 1)
-- total = price * qty (no aggregates)
INSERT INTO CusOrder (userID, totalAmount, orderStatus)
VALUES (1, @price * @qty, 'Processing');
SET @o_low_stock := LAST_INSERT_ID();

-- 1 line item
INSERT INTO OrderItem (orderID, productID, quantity, priceAtPurchase)
VALUES (@o_low_stock, @prod, @qty, @price);

-- pay exactly the total so only stock causes the error
INSERT INTO Payment (orderID, paymentMethodID, amountPaid, paymentStatus)
VALUES (@o_low_stock, @pm_card, @price * @qty, 'Completed');

-- expect: Error Code 1644 + 'Insufficient stock (BR5)'
CALL CheckoutOrder(@o_low_stock, 0);
```

#	Time	Action	Message
2	21:48:56	SET @pm_card := (SELECT paymentMethodID FROM PaymentMethod WHERE methodName IN ('Credit Card','Card') LIMIT 1)	0 row(s) affected
3	21:48:56	SET @prod := 11	0 row(s) affected
4	21:48:56	SET @price := (SELECT price FROM Product WHERE productID = @prod)	0 row(s) affected
5	21:48:56	SET @stock := (SELECT stockQuantity FROM Product WHERE productID = @prod)	0 row(s) affected
6	21:48:56	SET @qty := @stock + 1	0 row(s) affected
7	21:48:56	INSERT INTO CusOrder (userID, totalAmount, orderStatus) VALUES (1, @price * @qty, 'Processing')	1 row(s) affected
8	21:48:56	SET @o_low_stock := LAST_INSERT_ID()	0 row(s) affected
9	21:48:56	INSERT INTO OrderItem (orderID, productID, quantity, priceAtPurchase) VALUES (@o_low_stock, @prod, @qty, @price)	1 row(s) affected
10	21:48:56	INSERT INTO Payment (orderID, paymentMethodID, amountPaid, paymentStatus) VALUES (@o_low_stock, @pm_card, @price * @qty, 'Completed')	1 row(s) affected
11	21:48:56	CALL CheckoutOrder(@o_low_stock, 0)	Error Code: 1644. Insufficient stock (BR5)

Figure 3: BR5 Output.

BR2 – Payments Do Not Equal Order Total

```
-- pick a payment method
SET @pm_card := (
  SELECT paymentMethodID FROM PaymentMethod
  WHERE methodName IN ('Credit Card','Card')
  LIMIT 1
);

-- simple product (Cotton T-Shirt, id = 3)
SET @prod := 3;
SET @price := (SELECT price FROM Product WHERE productID = @prod);

-- user with primary address (userID = 1)
INSERT INTO CusOrder (userID, totalAmount, orderStatus)
VALUES (1, @price, 'Processing');
SET @o_bad_pay := LAST_INSERT_ID();

-- 1 line item
INSERT INTO OrderItem (orderID, productID, quantity, priceAtPurchase)
VALUES (@o_bad_pay, @prod, 1, @price);

-- pay LESS than total (so the sum of Completed/Approved != total)
INSERT INTO Payment (orderID, paymentMethodID, amountPaid, paymentStatus)
VALUES (@o_bad_pay, @pm_card, 1.00, 'Completed');  -- 1.00 < @price

-- (optional: add a 'Pending' payment; it won't be counted)
-- INSERT INTO Payment (orderID, paymentMethodID, amountPaid,
   paymentStatus)
-- VALUES (@o_bad_pay, @pm_card, @price - 1.00, 'Pending');

-- expect: Error Code 1644 + 'Payment total does not equal order total (
   BR2)'
CALL CheckoutOrder(@o_bad_pay, 0);
```



#	Time	Action	Message
1	21:49:39	USE COMP2350_zA2W29TeamD	0 row(s) affected
2	21:49:39	SET @pm_card := (SELECT paymentMethodID FROM PaymentMethod WHERE methodName IN ('Credit Card','Card') LIMIT 1)	0 row(s) affected
3	21:49:39	SET @prod := 3	0 row(s) affected
4	21:49:39	SET @price := (SELECT price FROM Product WHERE productID = @prod)	0 row(s) affected
5	21:49:39	INSERT INTO CusOrder (userID, totalAmount, orderStatus) VALUES (1, @price, 'Processing')	1 row(s) affected
6	21:49:39	SET @o_bad_pay := LAST_INSERT_ID()	0 row(s) affected
7	21:49:39	INSERT INTO OrderItem (orderID, productID, quantity, priceAtPurchase) VALUES (@o_bad_pay, @prod, 1, @price)	1 row(s) affected
8	21:49:39	INSERT INTO Payment (orderID, paymentMethodID, amountPaid, paymentStatus) VALUES (@o_bad_pay, @pm_card, 1.00, 'Completed')	1 row(s) affected
9	21:49:39	CALL CheckoutOrder(@o_bad_pay, 0)	Error Code: 1644. Payment total does not equal order total (BR2)

Figure 4: BR2 Output.

Success Example – Exact Payment & Small Points Redemption

```
-- pick a payment method
SET @pm_card := (
  SELECT paymentMethodID FROM PaymentMethod
  WHERE methodName IN ('Credit Card','Card')
  LIMIT 1
);

-- product with stock (use id = 3)
SET @prod := 3;
SET @price := (SELECT price FROM Product WHERE productID = @prod);

-- small points to redeem to keep it safe (most seeds give user 1 enough
   points)
SET @redeem := 10;

-- user with primary address (userID = 1)
-- total = 2 * price (no aggregates)
INSERT INTO CusOrder (userID, totalAmount, orderStatus)
VALUES (1, 2 * @price, 'Processing');
SET @o_ok := LAST_INSERT_ID();

-- 1 line item with quantity 2
INSERT INTO OrderItem (orderID, productID, quantity, priceAtPurchase)
VALUES (@o_ok, @prod, 2, @price);

-- pay the exact total (Completed)
INSERT INTO Payment (orderID, paymentMethodID, amountPaid, paymentStatus)
VALUES (@o_ok, @pm_card, 2 * @price, 'Completed');

-- expect: success (no error), stock deducted, points redeemed by 10
CALL CheckoutOrder(@o_ok, @redeem);

-- quick check
SELECT loyaltyPoints FROM `User` WHERE userID = 1;
SELECT productID, stockQuantity FROM Product WHERE productID = @prod;
```

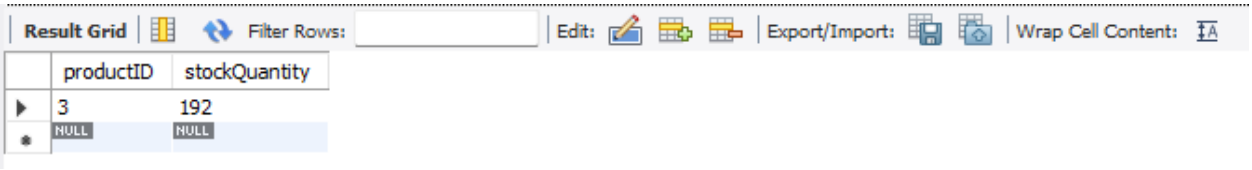


Figure 5: Success Output.

Section 4: Task 4.1 – Trigger Designs (Design Only)

Two trigger *designs* per the spec:

- **Stock Deduction (BR5)**: `BEFORE INSERT ON OrderItem`. If `Product.stockQuantity < NEW.quantity`, then `SIGNAL 'Insufficient stock (BR5)'`; otherwise decrement `Product.stockQuantity` by `NEW.quantity`.
- **Overpayment Prevention (BR2)**: `BEFORE INSERT ON Payment` (and/or `BEFORE UPDATE`). Compute sum of `Completed/Approved` + `NEW.amountPaid` (if `NEW` qualifies). If it would exceed `CusOrder.totalAmount`, then `SIGNAL 'Overpayment not allowed (BR2)'`.

Section 5: Task 4.3 – Trigger Testing (Implemented Triggers)

The following SQL drives the *implemented* triggers:

- **Refund Processing (BR7/BR8)**: When a returned item is accepted, insert one `Refund` capped at original purchase amount.
- **Loyalty Points Adjustment (BR6/BR10)**: Credit points on `Delivered`; claw back on `Cancelled/Returned`; avoid double-counting.

Test SQL:

```
-- Refund flow using the success order's item (caps refund to purchase
   price)
INSERT INTO ReturnedItem(orderItemID, returnReason, requestedAt,
   returnStatus)
VALUES (1, 'Beginner test', NOW(), 'Pending');
UPDATE ReturnedItem
   SET returnStatus='Accepted', decisionDate=NOW(), refundAmount=9999.99
 WHERE returnID = LAST_INSERT_ID();
SELECT * FROM Refund ORDER BY refundID DESC LIMIT 3;

-- Loyalty: mark the success order Delivered to credit points, then
   Cancelled to claw back
UPDATE CusOrder SET orderStatus='Delivered' WHERE orderID = 2;
SELECT u.loyaltyPoints, lt.*
  FROM `User` u JOIN LoyaltyTransaction lt ON u.userID = lt.userID
```

```
 WHERE lt.orderID = 2;
UPDATE CusOrder SET orderStatus='Cancelled' WHERE orderID = 2;
SELECT u.loyaltyPoints, lt.*
  FROM `User` u JOIN LoyaltyTransaction lt ON u.userID = lt.userID
 WHERE lt.orderID = 2;
```

refundID	returnID	refundMethod	refundAmount	processedAt
7	9	Original Method	299.99	2025-10-23 18:21:57
6	8	Original Method	299.99	2025-10-23 18:11:48
5	7	Original Method	299.99	2025-10-23 18:10:59
NULL	NULL	NULL	NULL	NULL

Output for Refund Testing

loyaltyPoints	transactionID	userID	orderID	pointsEarned	pointsSpent	transactionDate
780	12	3	2	239	0	2025-10-23 18:10:51
780	13	3	2	0	239	2025-10-23 18:10:51

Output for Loyalty Points Testing

Figure 6: Task 4.3 Output