CSE 414 PROJECT REPORT

DATABASE OF ONLINE GAME PLATFORM

EMRE SEZER

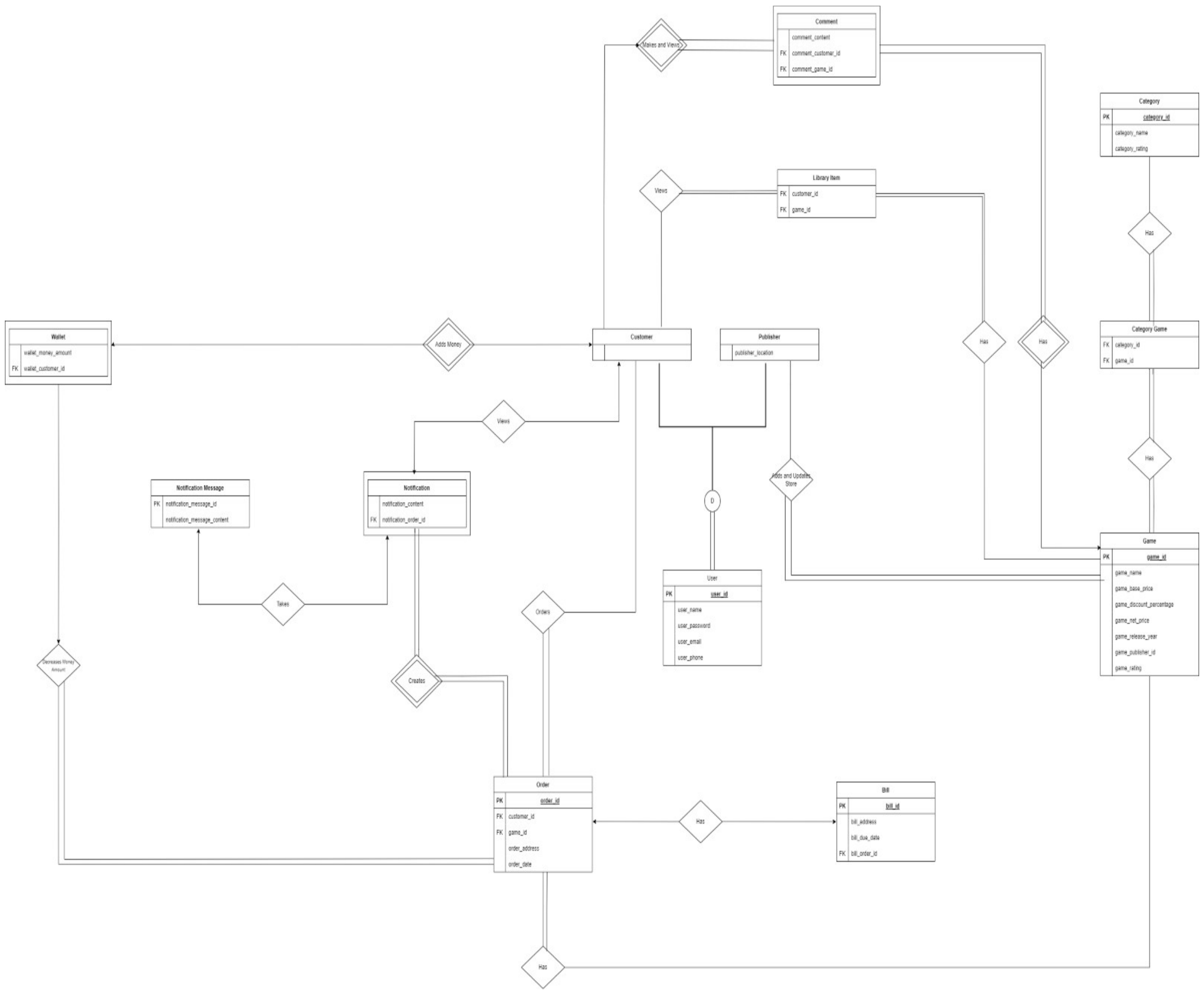
1901042640

User Requirements:

- Publisher must be able to add their games to the store
- Publisher must be able to update their games price and discount percentage on the store
- Customers can view their games at their library
- Customers should be able to buy games from store
- Customers can view their order informations
- Customers can view their bill informations
- Customers should be able to add money to their wallet
- Customers can sort the games on the store with multiple options
- Customers can filter their games at their library depending on category of the game
- Customers can comment on games
- Customers can view comments on a game previously done by the customers
- Customers can view notifications

ER Diagram:

I added ER diagram of the project additionally. You can view it, since it might be hard to read the texts on the image above.



Functional Dependencies:

Games:
game_id -> game_name, game_base_price, game_discount_percentage, game_net_price, game_release_year, game_publisher_id, game_rating

game_rating -> game_name, game_base_price, game_discount_percentage, game_net_price, game_release_year, game_publisher_id, game_id

Category:
category_id -> category_name, category_rating

category_rating -> category_name, category_id

Category Game:
category_id -> game_id

game_id -> category_id

Customer:
user-id -> user_name, user_email, user_phone, user_password

user_email -> user_name, user_id, user_phone, user_password

Publisher:
user-id -> user_name, user_email, user_phone, user_password, publisher_location

user_email -> user_name, user_id, user_phone, user_password, publisher_location

Library Item:

game_id -> customer_id

customer_id -> game_id

Wallet:

wallet_customer -> wallet_money_amount

Notification:

notification_order_id -> notification_content

Notification Message:

notification_message_id -> notification_message_content

Order:

order_id -> customer_id, game_id, order_address, order_date

Bill:

bill_id -> bill_address, bill_due_date, bill_order_id

bill_order_id -> bill_address, bill_due_date, bill_id

Comment:

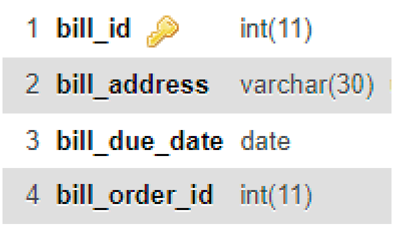
comment_game_id, comment_game_id -> comment_content, comment_customer_id, comment_content

Normalization:

Normalization is applied to the tables.
All Id’s are AUTO_INCREMENT. Each id is primary key or foreign key. If id is primary key then it is candidate key or super key.
If id is foreign key then it is candidate key or primary key with another foreign key.
All of the tables are in BCNF.

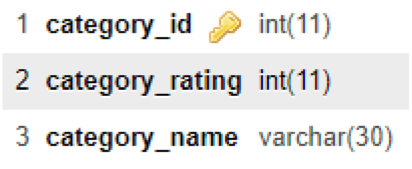
Tables:

bills:



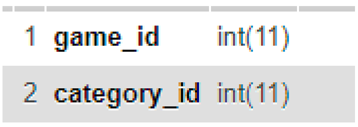
1	bill_id	🔑	int(11)
2	bill_address		varchar(30)
3	bill_due_date		date
4	bill_order_id		int(11)

categories:



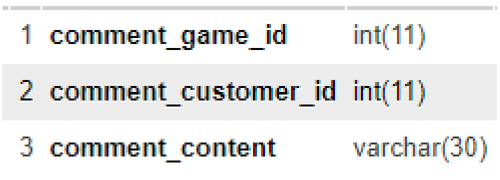
1	category_id	🔑	int(11)
2	category_rating		int(11)
3	category_name		varchar(30)

category_and_games:



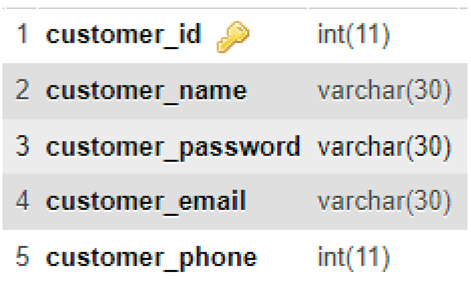
1	game_id	int(11)
2	category_id	int(11)

comments:



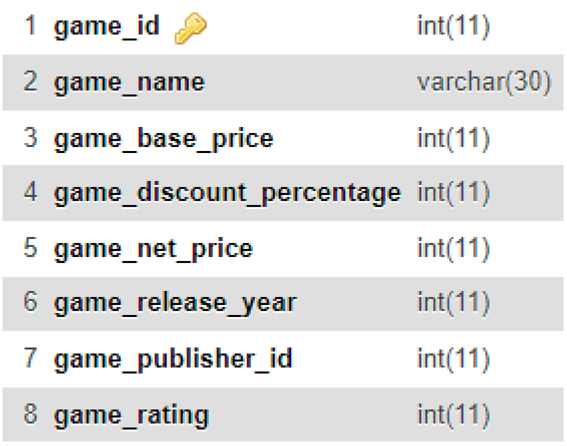
1	comment_game_id	int(11)
2	comment_customer_id	int(11)
3	comment_content	varchar(30)

customers:



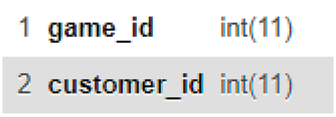
1	customer_id	🔑	int(11)
2	customer_name		varchar(30)
3	customer_password		varchar(30)
4	customer_email		varchar(30)
5	customer_phone		int(11)

games:



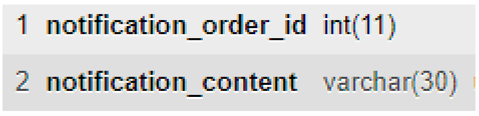
1	game_id	🔑	int(11)
2	game_name		varchar(30)
3	game_base_price		int(11)
4	game_discount_percentage		int(11)
5	game_net_price		int(11)
6	game_release_year		int(11)
7	game_publisher_id		int(11)
8	game_rating		int(11)

library_items:



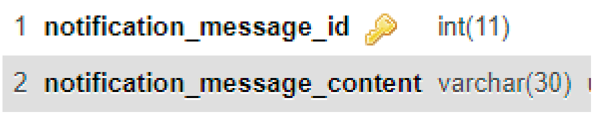
1	game_id	int(11)
2	customer_id	int(11)

notifications:



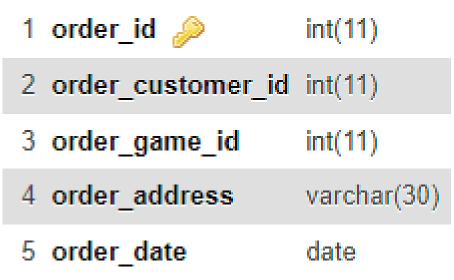
1	notification_order_id	int(11)
2	notification_content	varchar(30)

notification_messages:



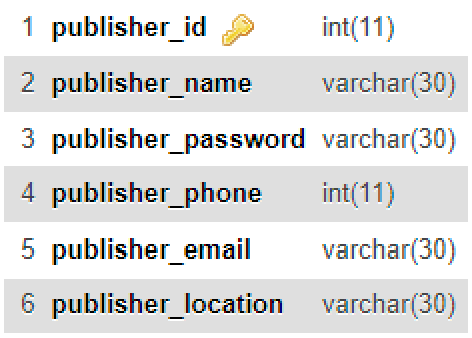
1	notification_message_id	🔑	int(11)
2	notification_message_content		varchar(30)

orders:



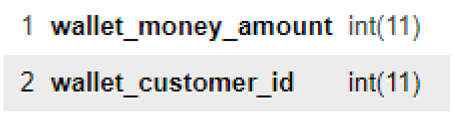
1	order_id	🔑	int(11)
2	order_customer_id		int(11)
3	order_game_id		int(11)
4	order_address		varchar(30)
5	order_date		date

publishers:



1	publisher_id 🔑	int(11)
2	publisher_name	varchar(30)
3	publisher_password	varchar(30)
4	publisher_phone	int(11)
5	publisher_email	varchar(30)
6	publisher_location	varchar(30)

wallets:



1	wallet_money_amount	int(11)
2	wallet_customer_id	int(11)

Weak entities are wallets, notifications, comments

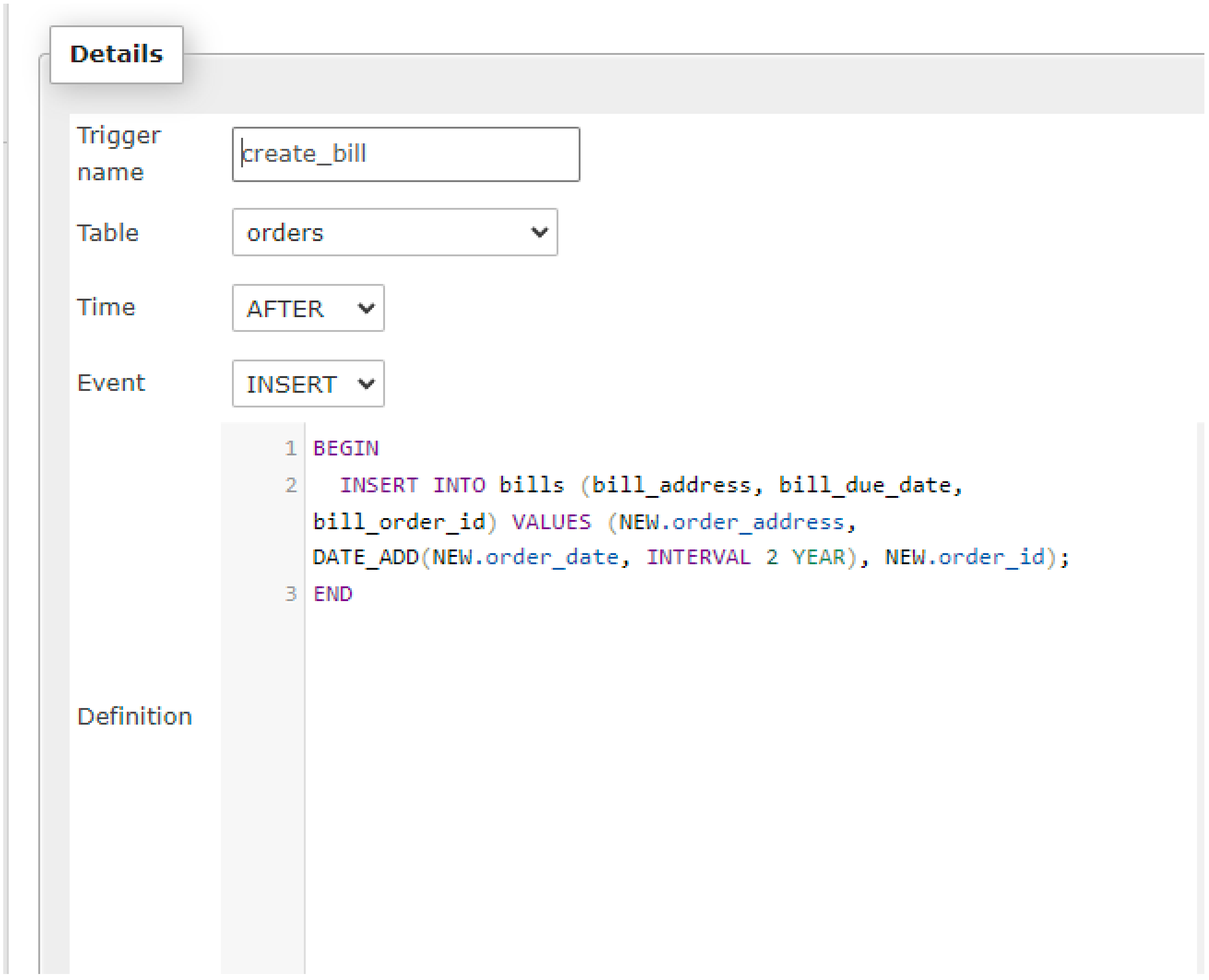
Triggers:

Name:

create_bill

Definition:

After an order is inserted into orders a new bill is inserted into bills



Trigger name	create_bill
Table	orders
Time	AFTER
Event	INSERT
Definition	

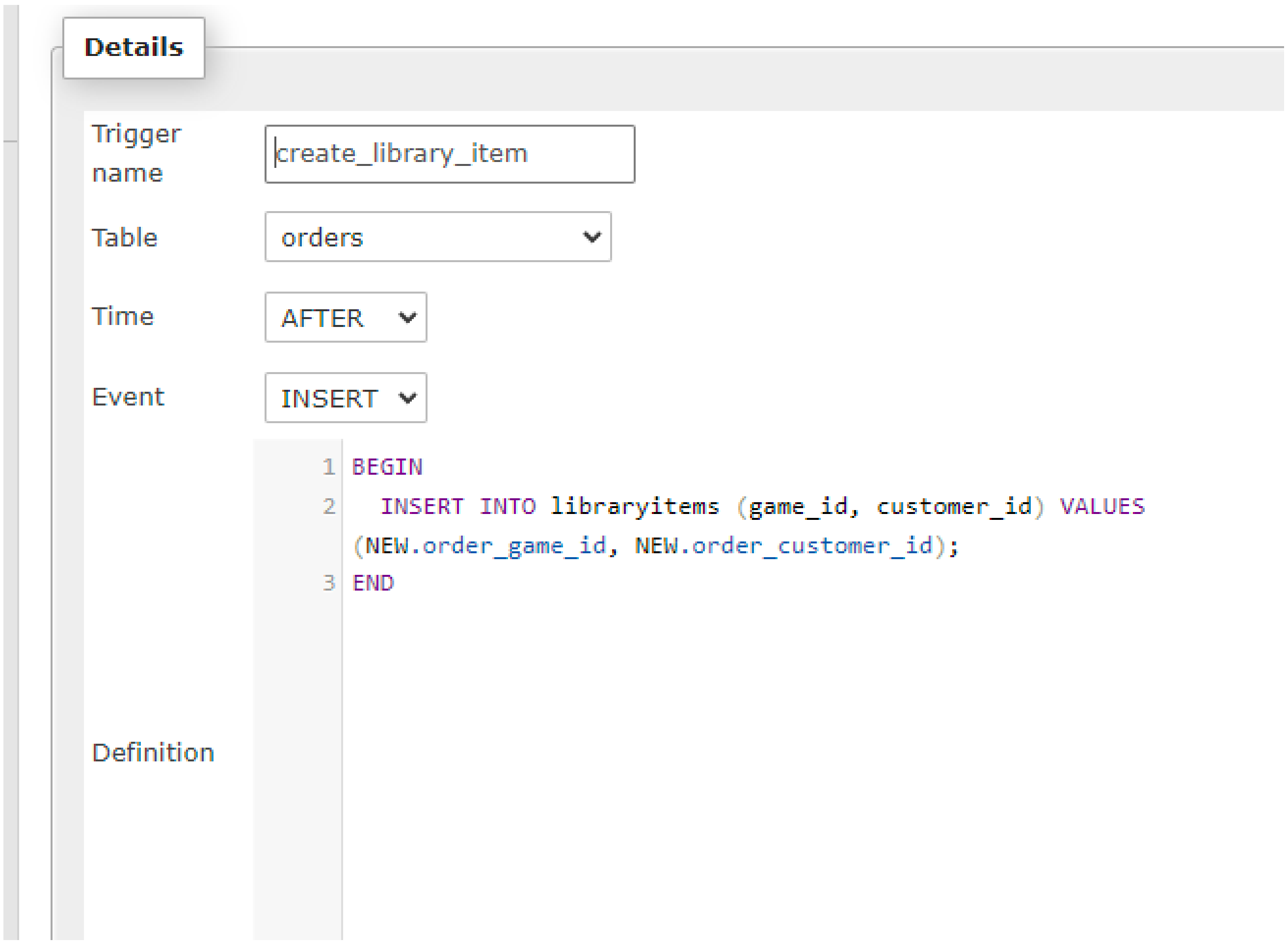
```
BEGIN
  INSERT INTO bills (bill_address, bill_due_date,
bill_order_id) VALUES (NEW.order_address,
DATE_ADD(NEW.order_date, INTERVAL 2 YEAR), NEW.order_id);
END
```

Name:

create_library_item

Definition:

After an order is inserted into orders a new library item is inserted into library_items



Details

Trigger name	create_library_item
Table	orders
Time	AFTER
Event	INSERT
Definition	

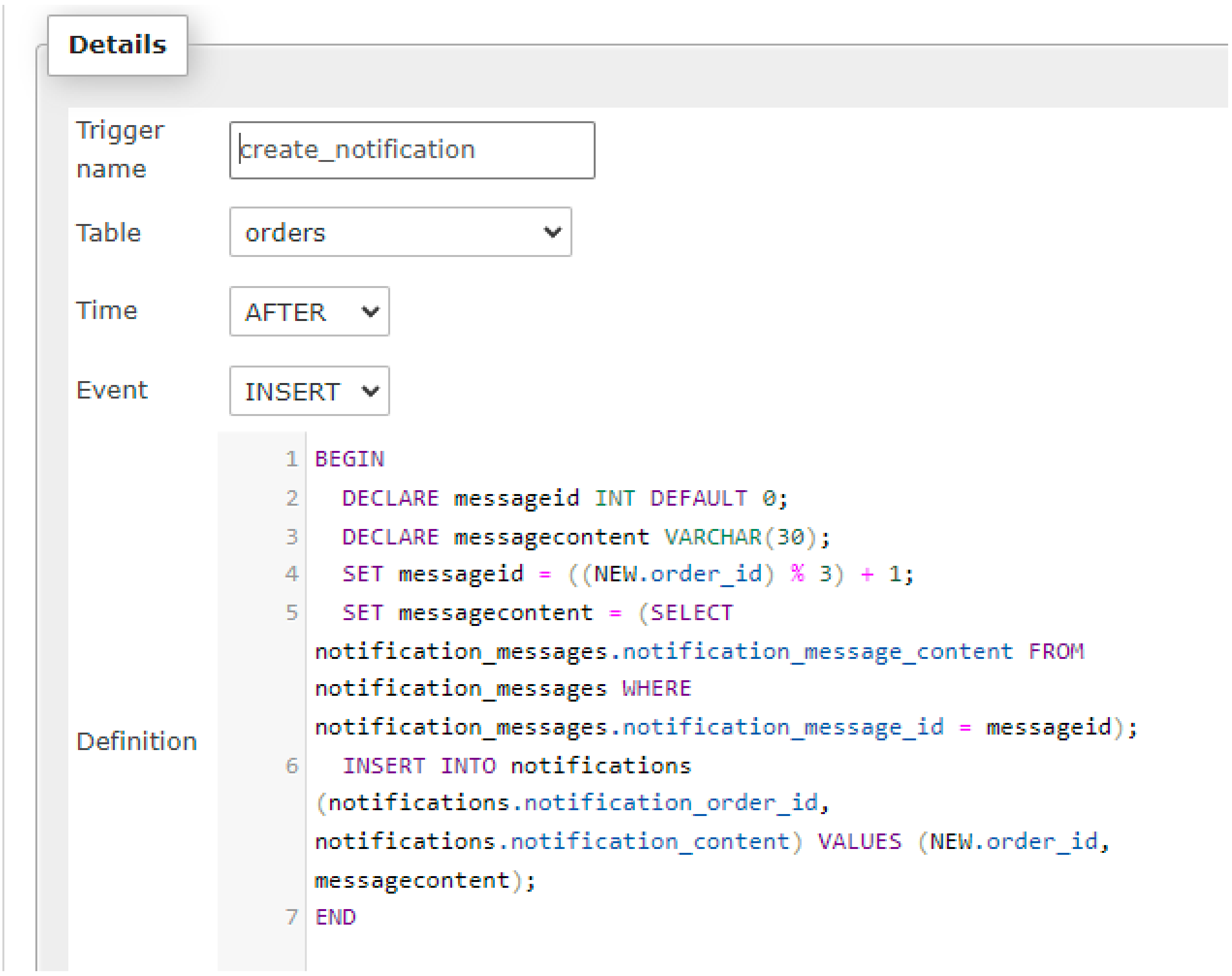
```
BEGIN
  INSERT INTO libraryitems (game_id, customer_id) VALUES
(NEW.order_game_id, NEW.order_customer_id);
END
```

Name:

create_notification

Definition:

After an order is inserted into orders a notification is inserted into notifications



Details

Trigger name	create_notification
Table	orders
Time	AFTER
Event	INSERT

Definition

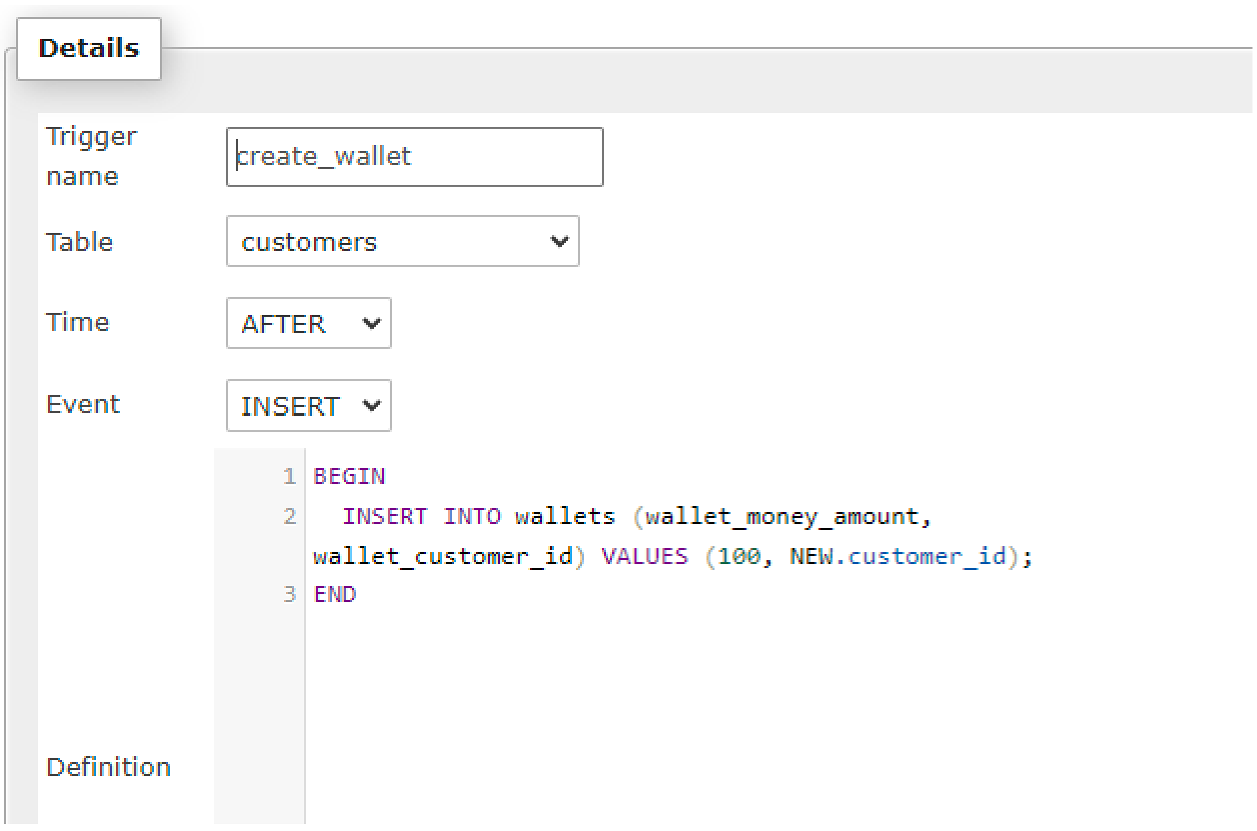
```
BEGIN
  DECLARE messageid INT DEFAULT 0;
  DECLARE messagecontent VARCHAR(30);
  SET messageid = ((NEW.order_id) % 3) + 1;
  SET messagecontent = (SELECT notification_messages.notification_message_content FROM notification_messages WHERE notification_messages.notification_message_id = messageid);
  INSERT INTO notifications (notifications.notification_order_id, notifications.notification_content) VALUES (NEW.order_id, messagecontent);
END
```

Name:

create_wallet

Definition:

After a customer is inserted into customers a new wallet is inserted into wallets



Details

Trigger name	create_wallet
Table	customers
Time	AFTER
Event	INSERT
Definition	

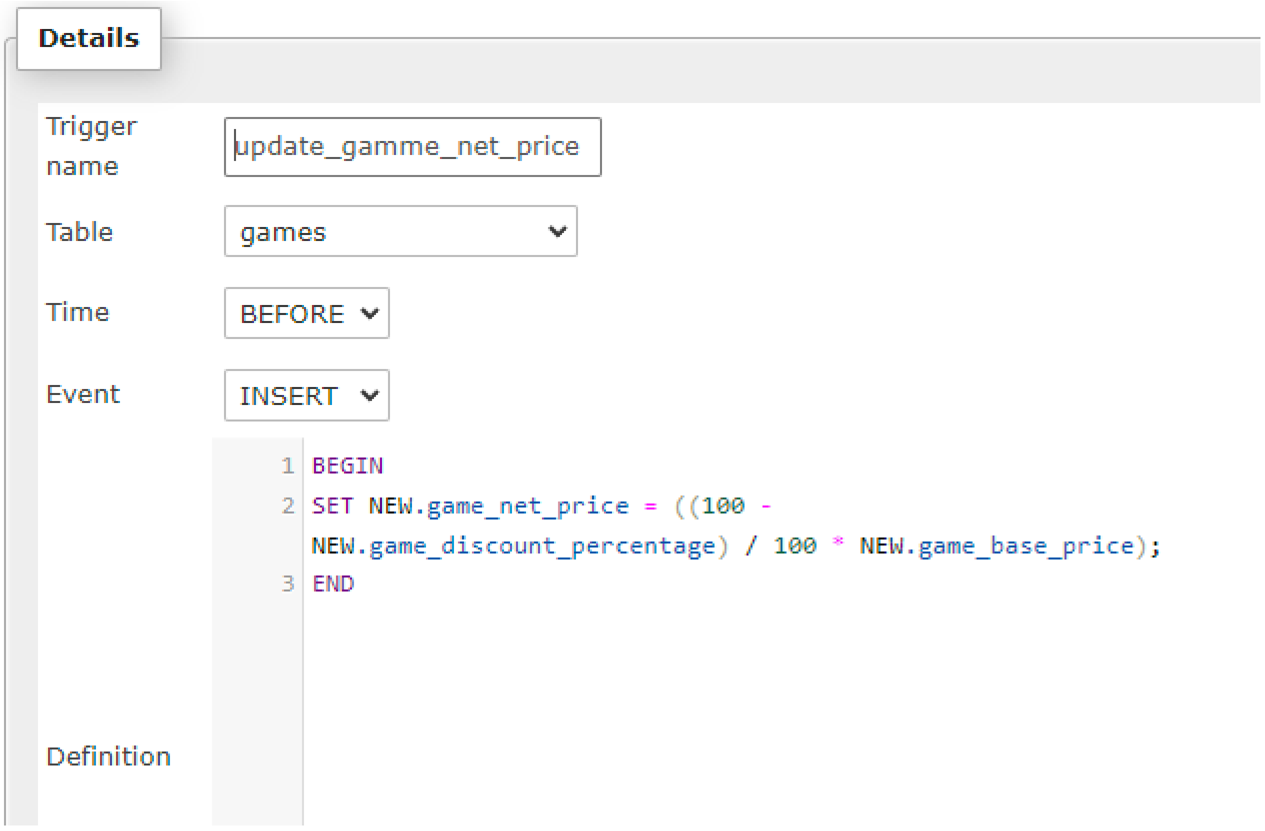
```
BEGIN
  INSERT INTO wallets (wallet_money_amount,
wallet_customer_id) VALUES (100, NEW.customer_id);
END
```

Name:

update_game_net_price

Definition:

Before a game is inserted into games game net price is set



Details

Trigger name	update_gamme_net_price
Table	games
Time	BEFORE
Event	INSERT
Definition	

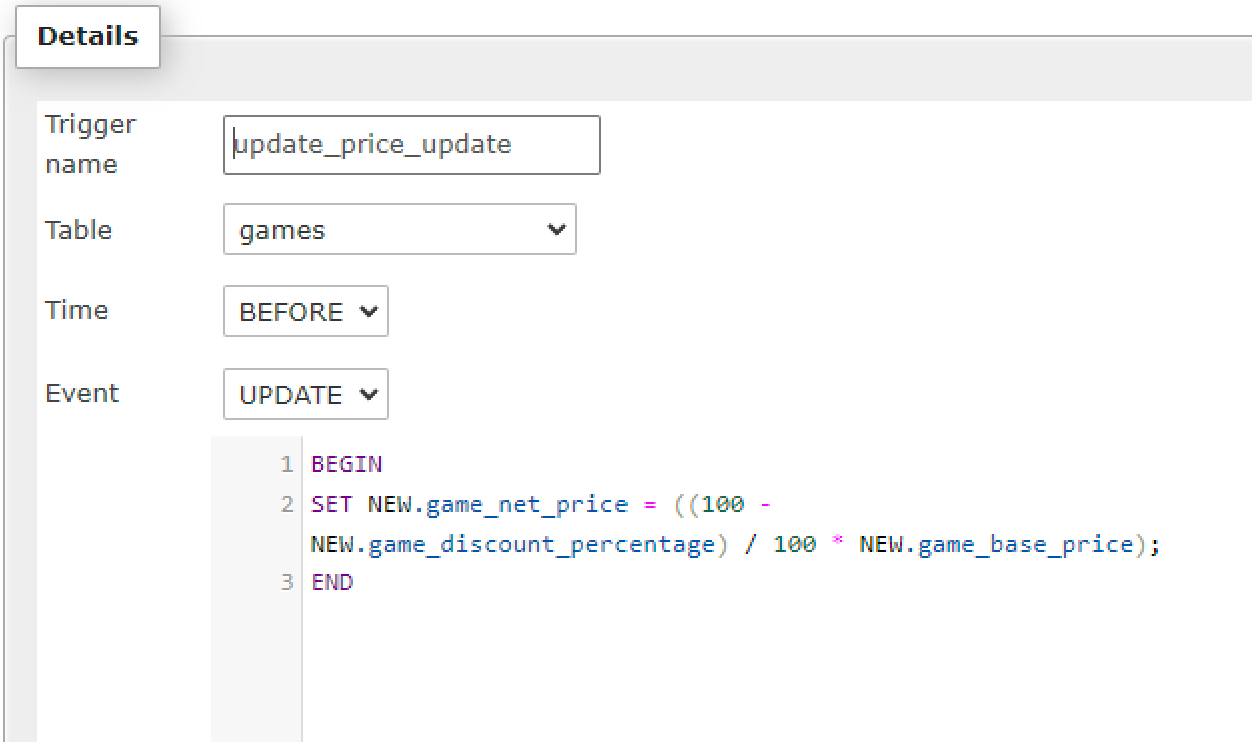
```
BEGIN
SET NEW.game_net_price = ((100 -
NEW.game_discount_percentage) / 100 * NEW.game_base_price);
END
```

Name:

update_price_update

Definition:

Before a game is updated in games game net price is set



Trigger name	update_price_update
Table	games
Time	BEFORE
Event	UPDATE

```
BEGIN
SET NEW.game_net_price = ((100 -
NEW.game_discount_percentage) / 100 * NEW.game_base_price);
END
```

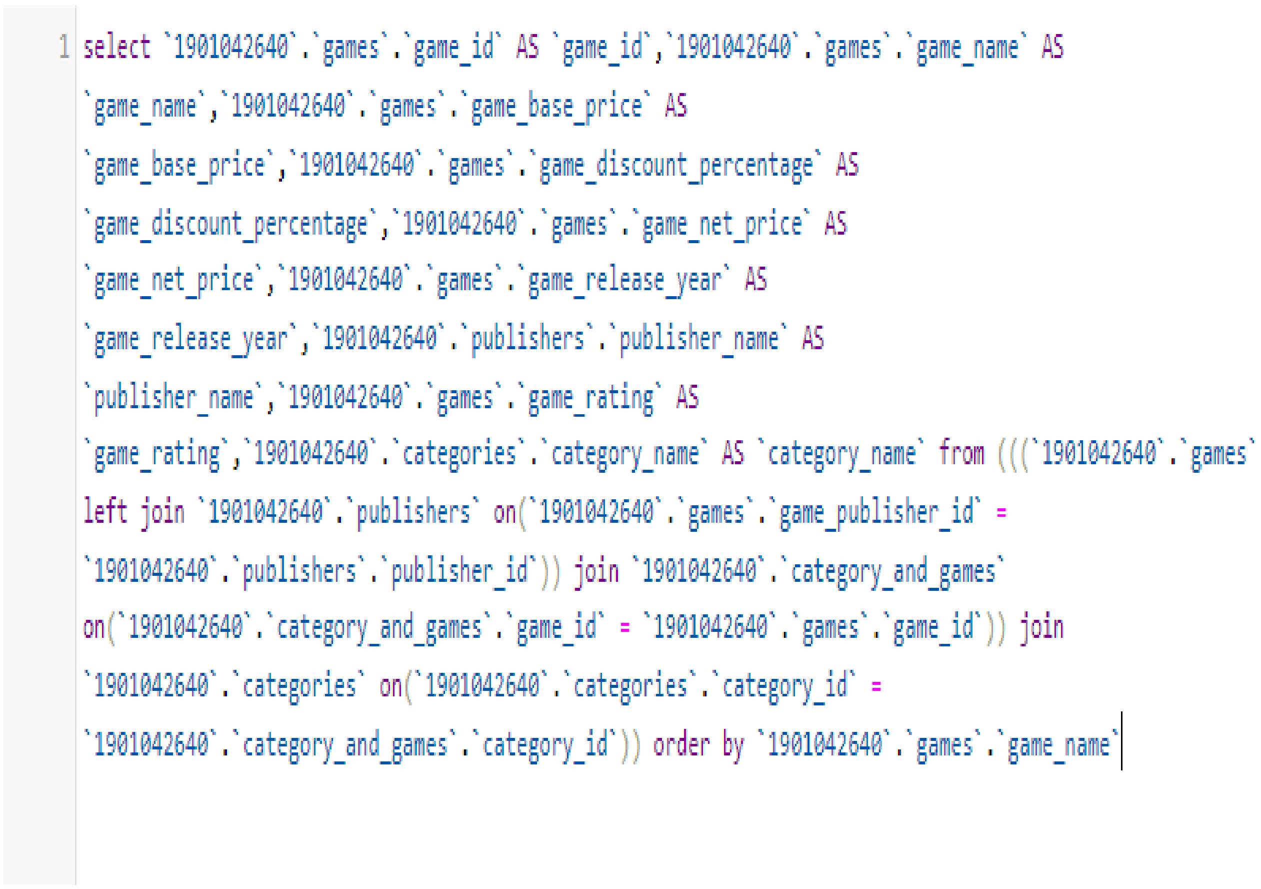
Views:

Name:

sort_games_by_name

Definition:

Sorts games in games by ascending name order. Uses LEFT and INNER JOINs.



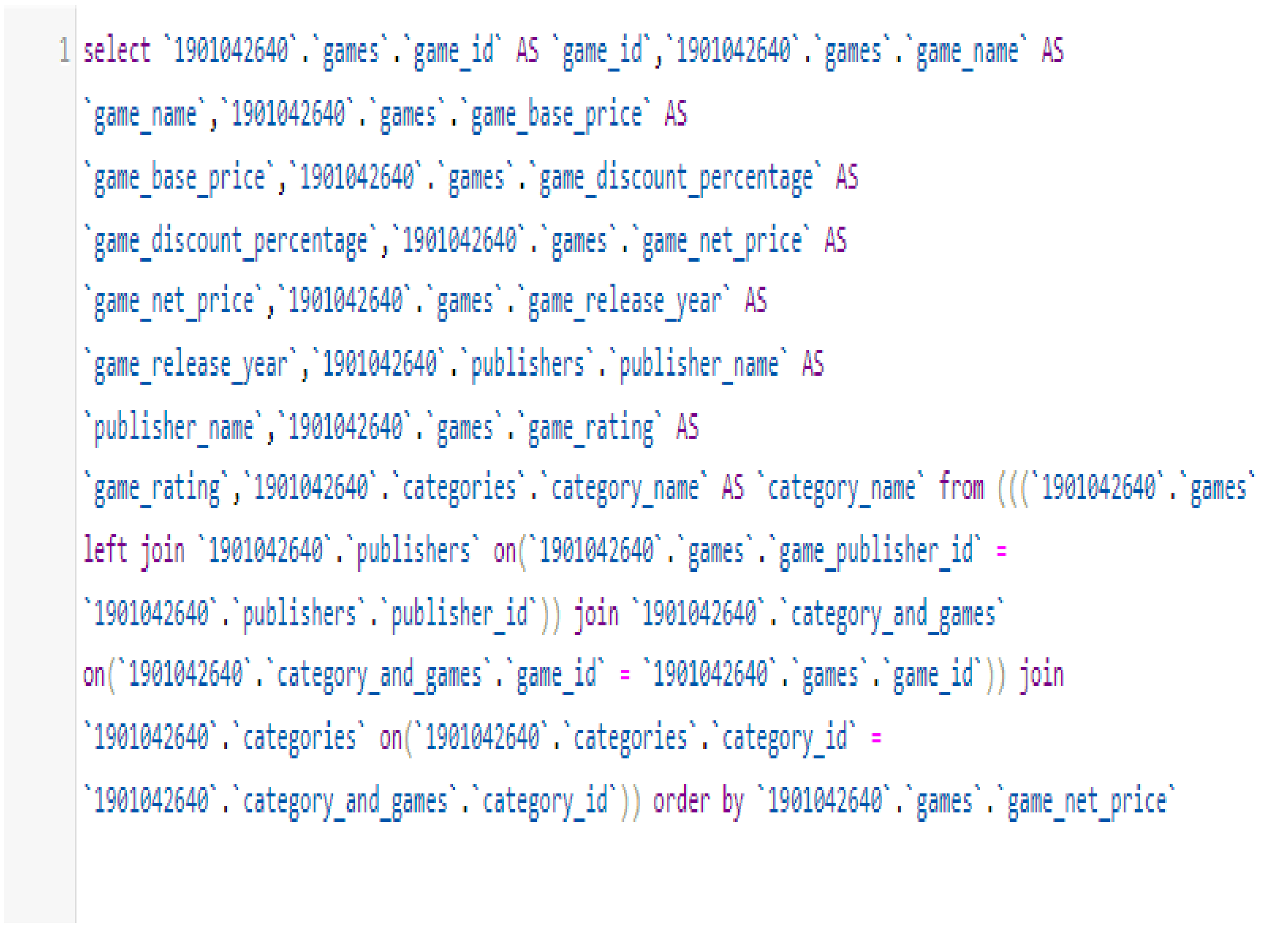
```
select `1901042640`.`games`.`game_id` AS `game_id`,`1901042640`.`games`.`game_name` AS
`game_name`,`1901042640`.`games`.`game_base_price` AS
`game_base_price`,`1901042640`.`games`.`game_discount_percentage` AS
`game_discount_percentage`,`1901042640`.`games`.`game_net_price` AS
`game_net_price`,`1901042640`.`games`.`game_release_year` AS
`game_release_year`,`1901042640`.`publishers`.`publisher_name` AS
`publisher_name`,`1901042640`.`games`.`game_rating` AS
`game_rating`,`1901042640`.`categories`.`category_name` AS `category_name` from (((`1901042640`.`games`
left join `1901042640`.`publishers` on(`1901042640`.`games`.`game_publisher_id` =
`1901042640`.`publishers`.`publisher_id`)) join `1901042640`.`category_and_games`
on(`1901042640`.`category_and_games`.`game_id` = `1901042640`.`games`.`game_id`)) join
`1901042640`.`categories` on(`1901042640`.`categories`.`category_id` =
`1901042640`.`category_and_games`.`category_id`)) order by `1901042640`.`games`.`game_name`
```

Name:

sort_games_by_price

Definition:

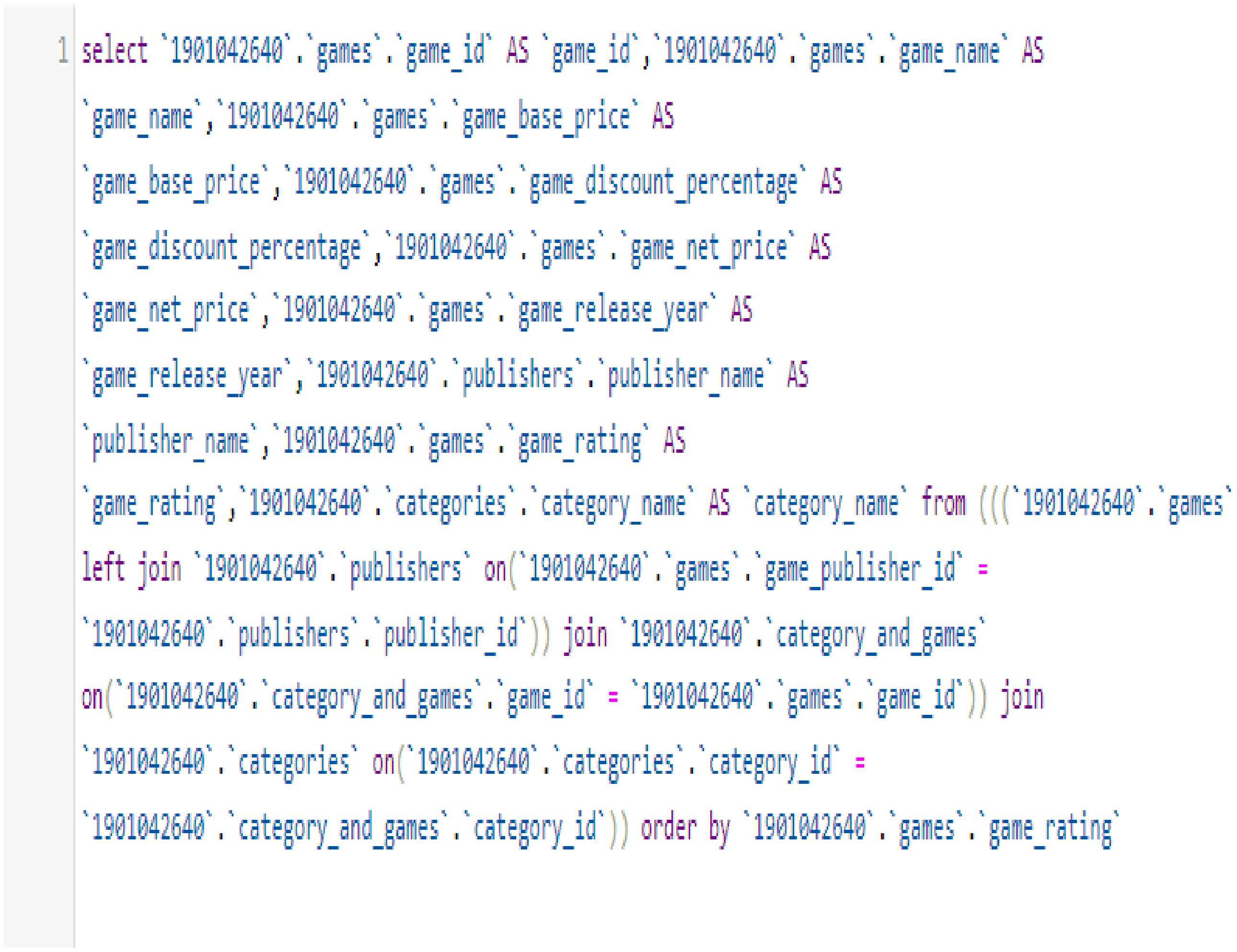
Sorts games in games by ascending net_price order. USES LEFT and INNER JOINs.



```
select `1901042640`.`games`.`game_id` AS `game_id`,`1901042640`.`games`.`game_name` AS
`game_name`,`1901042640`.`games`.`game_base_price` AS
`game_base_price`,`1901042640`.`games`.`game_discount_percentage` AS
`game_discount_percentage`,`1901042640`.`games`.`game_net_price` AS
`game_net_price`,`1901042640`.`games`.`game_release_year` AS
`game_release_year`,`1901042640`.`publishers`.`publisher_name` AS
`publisher_name`,`1901042640`.`games`.`game_rating` AS
`game_rating`,`1901042640`.`categories`.`category_name` AS `category_name` from (((`1901042640`.`games`
left join `1901042640`.`publishers` on(`1901042640`.`games`.`game_publisher_id` =
`1901042640`.`publishers`.`publisher_id`)) join `1901042640`.`category_and_games`
on(`1901042640`.`category_and_games`.`game_id` = `1901042640`.`games`.`game_id`)) join
`1901042640`.`categories` on(`1901042640`.`categories`.`category_id` =
`1901042640`.`category_and_games`.`category_id`)) order by `1901042640`.`games`.`game_net_price`
```

Name:
sort_games_by_rating

Definition:
Sorts games in games by descending ratingarder. Uses LEFT and INNER JOINs.



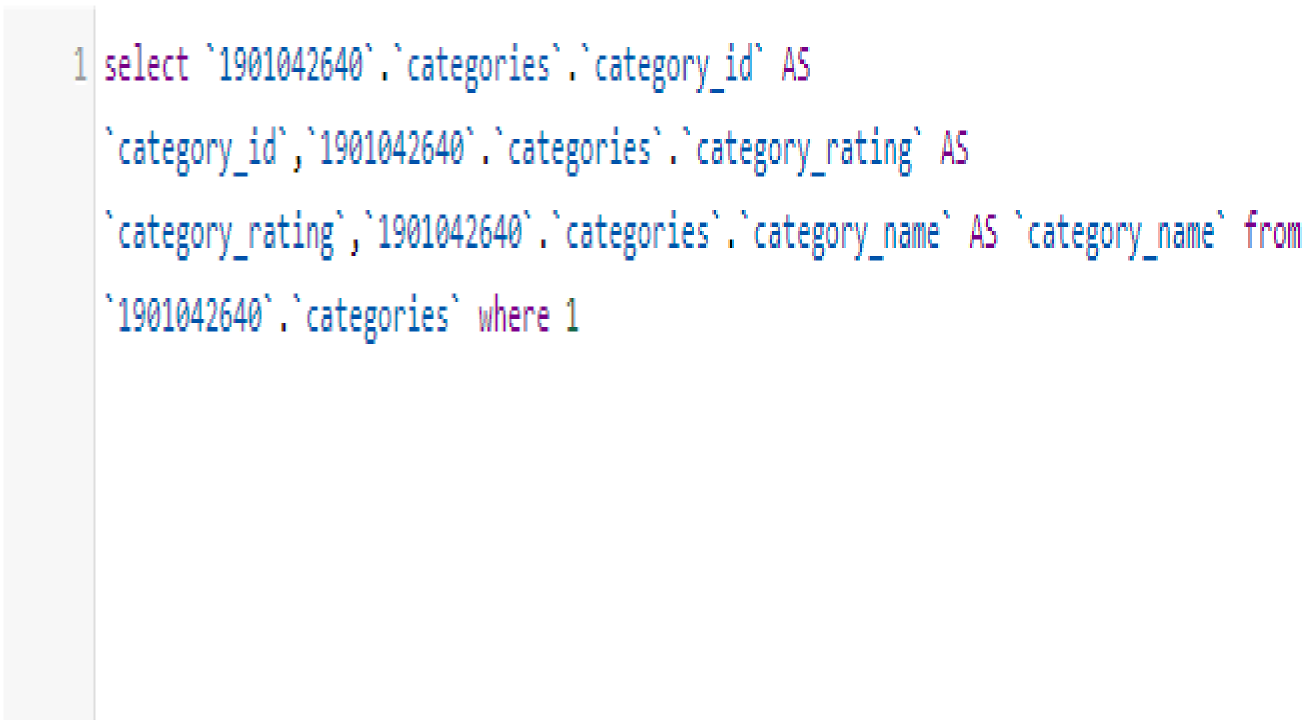
```sql
select `1901042640`.`games`.`game_id` AS `game_id`,`1901042640`.`games`.`game_name` AS
`game_name`,`1901042640`.`games`.`game_base_price` AS
`game_base_price`,`1901042640`.`games`.`game_discount_percentage` AS
`game_discount_percentage`,`1901042640`.`games`.`game_net_price` AS
`game_net_price`,`1901042640`.`games`.`game_release_year` AS
`game_release_year`,`1901042640`.`publishers`.`publisher_name` AS
`publisher_name`,`1901042640`.`games`.`game_rating` AS
`game_rating`,`1901042640`.`categories`.`category_name` AS `category_name` from (((`1901042640`.`games`
left join `1901042640`.`publishers` on(`1901042640`.`games`.`game_publisher_id` =
`1901042640`.`publishers`.`publisher_id`)) join `1901042640`.`category_and_games`
on(`1901042640`.`category_and_games`.`game_id` = `1901042640`.`games`.`game_id`)) join
`1901042640`.`categories` on(`1901042640`.`categories`.`category_id` =
`1901042640`.`category_and_games`.`category_id`)) order by `1901042640`.`games`.`game_rating`
```

Name:

view_all_categories

Definition:

Returns a table with all categories in categories.



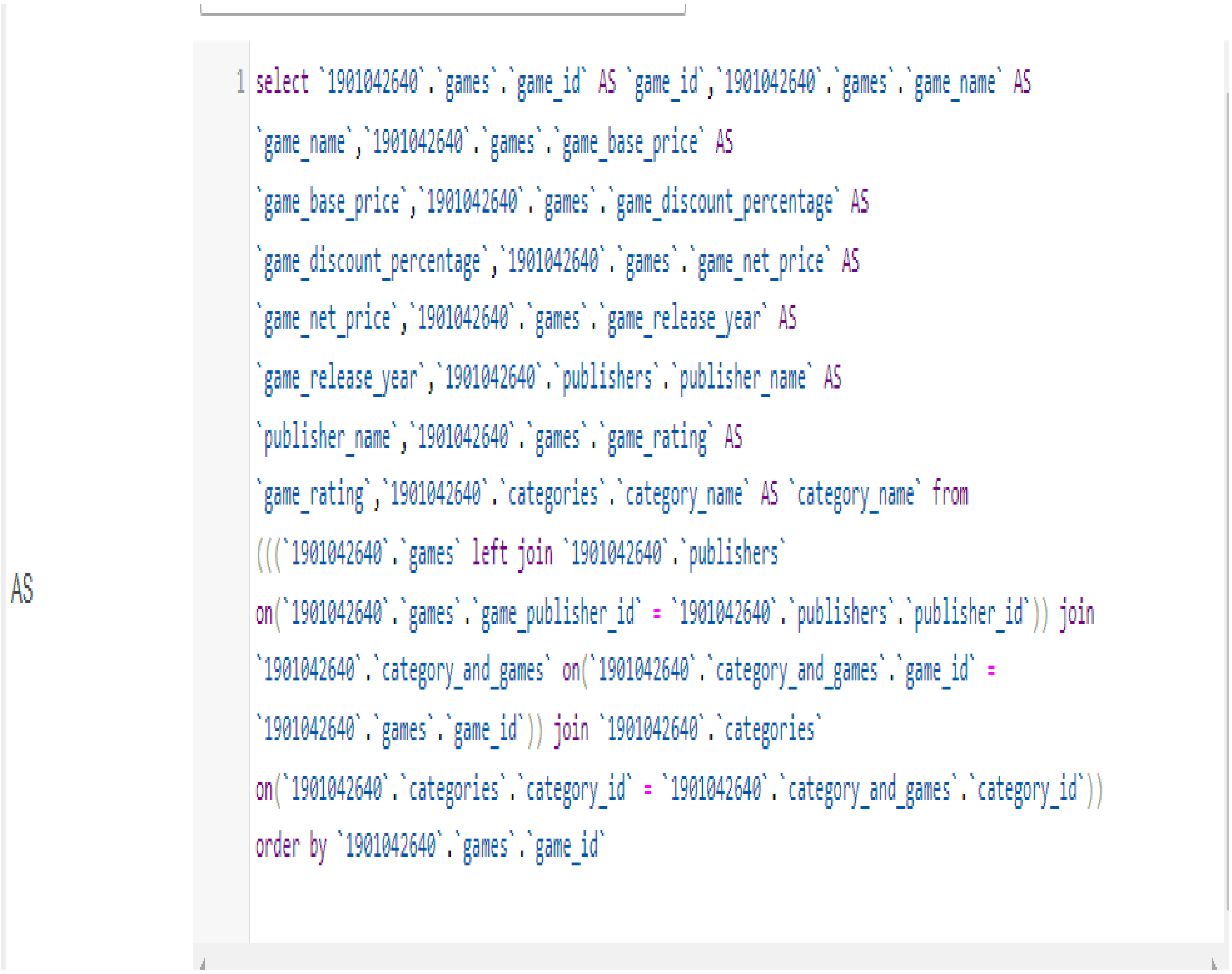
```
select `1901042640`.`categories`.`category_id` AS
`category_id`,`1901042640`.`categories`.`category_rating` AS
`category_rating`,`1901042640`.`categories`.`category_name` AS `category_name` from
`1901042640`.`categories` where 1
```

Name:

view_all_games

Definition:

Returns all games in games. Uses LEFT and INNER JOINs.



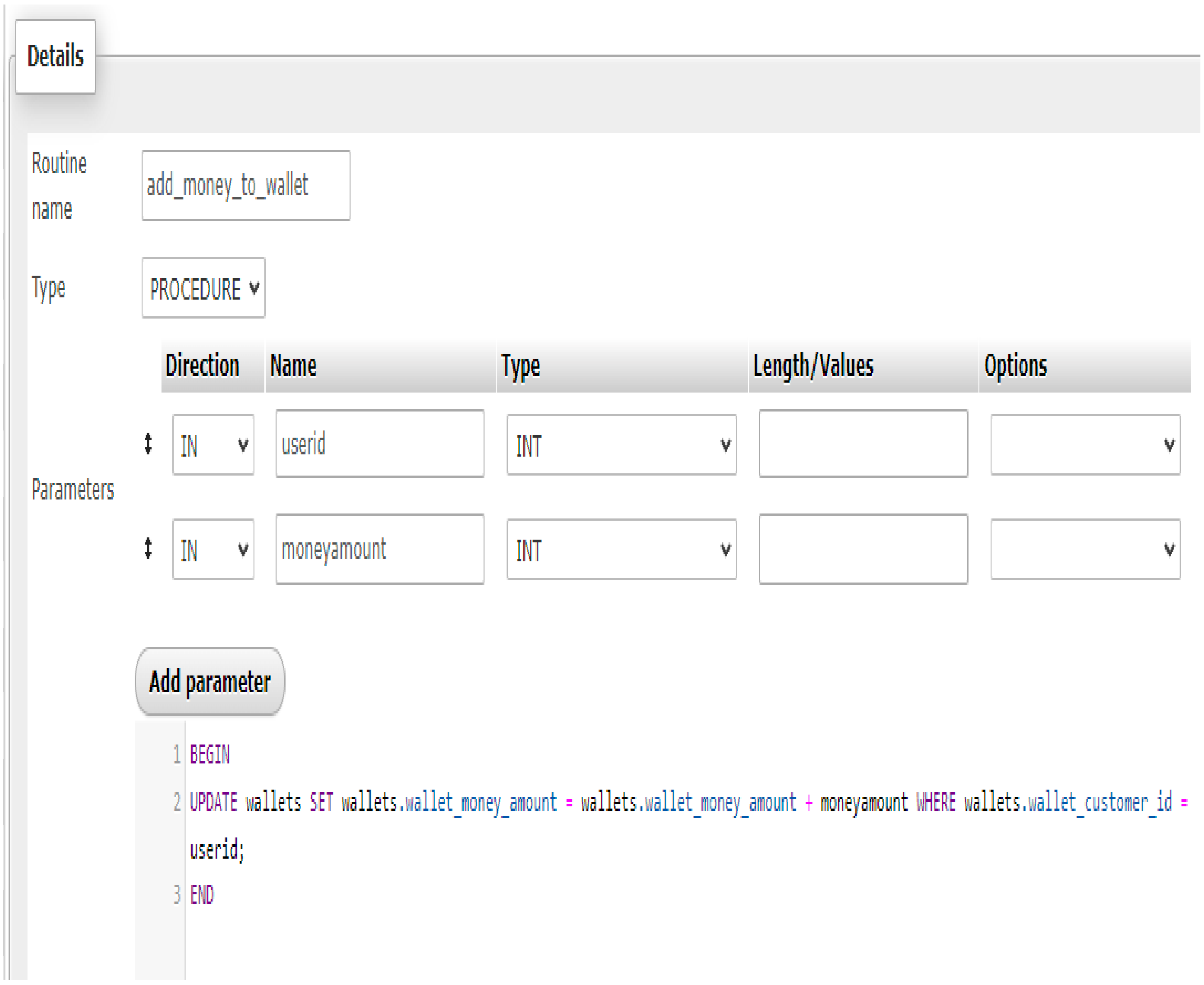
AS

```
select `1901042640`.`games`.`game_id` AS `game_id`,`1901042640`.`games`.`game_name` AS
`game_name`,`1901042640`.`games`.`game_base_price` AS
`game_base_price`,`1901042640`.`games`.`game_discount_percentage` AS
`game_discount_percentage`,`1901042640`.`games`.`game_net_price` AS
`game_net_price`,`1901042640`.`games`.`game_release_year` AS
`game_release_year`,`1901042640`.`publishers`.`publisher_name` AS
`publisher_name`,`1901042640`.`games`.`game_rating` AS
`game_rating`,`1901042640`.`categories`.`category_name` AS `category_name` from
(((`1901042640`.`games` left join `1901042640`.`publishers`
on(`1901042640`.`games`.`game_publisher_id` = `1901042640`.`publishers`.`publisher_id`)) join
`1901042640`.`category_and_games` on(`1901042640`.`category_and_games`.`game_id` =
`1901042640`.`games`.`game_id`)) join `1901042640`.`categories`
on(`1901042640`.`categories`.`category_id` = `1901042640`.`category_and_games`.`category_id`))
order by `1901042640`.`games`.`game_id`
```

Procedures:

Name:
add_money_to_wallet

Definition:
Takes userid and moneyamount as inputs. Adds the moneyemount to moneyamount of wallet where wallets.wallet_customer_id = userid.



Routine name	add_money_to_wallet
Type	PROCEDURE

Parameters

Direction	Name	Type	Length/Values	Options
IN	userid	INT		
IN	moneyamount	INT		

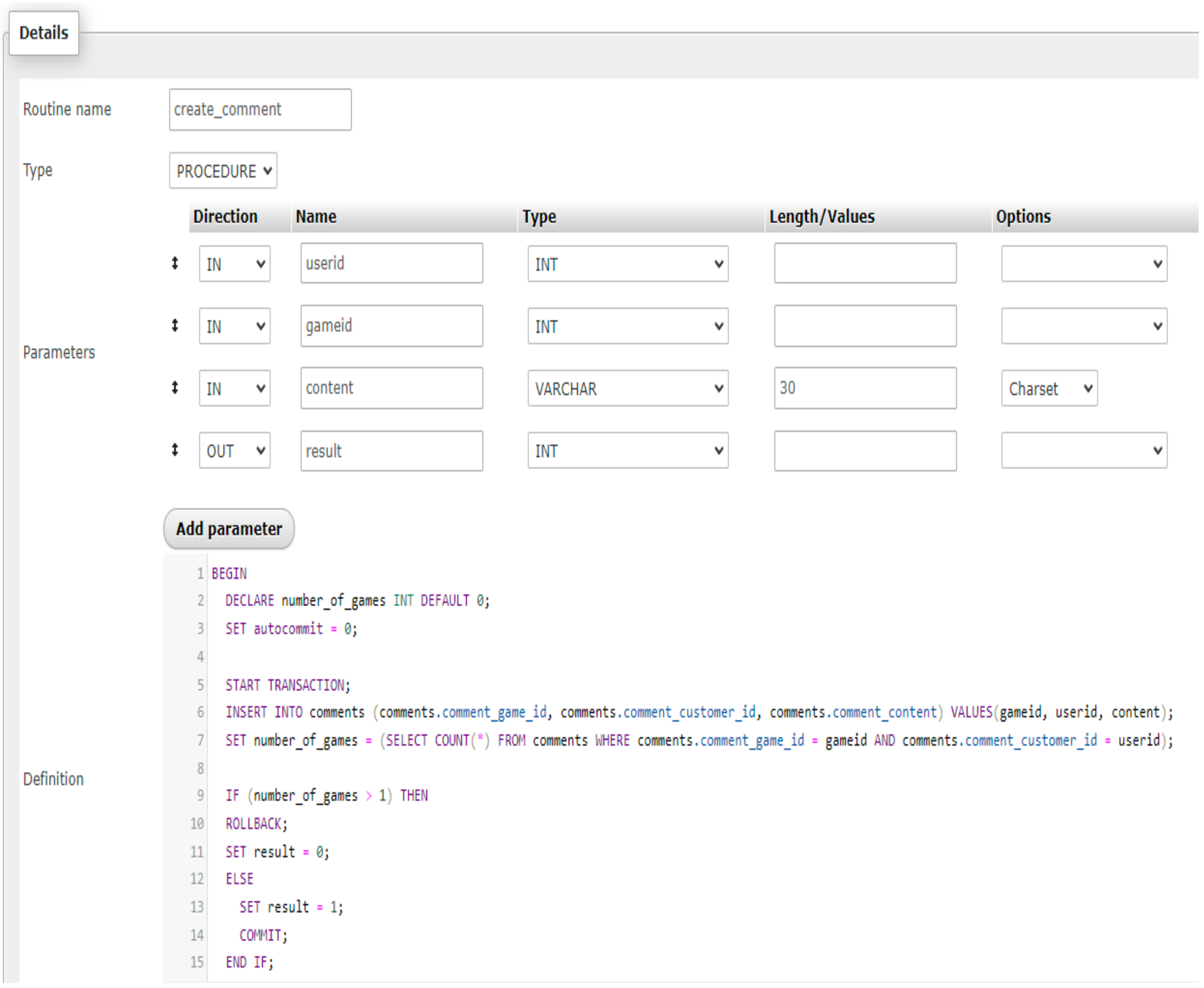
```
BEGIN
UPDATE wallets SET wallets.wallet_money_amount = wallets.wallet_money_amount + moneyamount WHERE wallets.wallet_customer_id =
userid;
END
```

Name:

create_comment

Definition:

Takes userid, gameid, content as inputs and returns result as output.
It starts an atomic transaction. Inserts into comments where
comment_customer_id = userid AND comment_game_id = gameid.
If there are more than 1 comments with same comment_game_id and
comment_customer_id then it rollbacks, else commits.



Routine name: create_comment

Type: PROCEDURE

Parameters:

Direction	Name	Type	Length/Values	Options
IN	userid	INT		
IN	gameid	INT		
IN	content	VARCHAR	30	Charset
OUT	result	INT		

Definition:

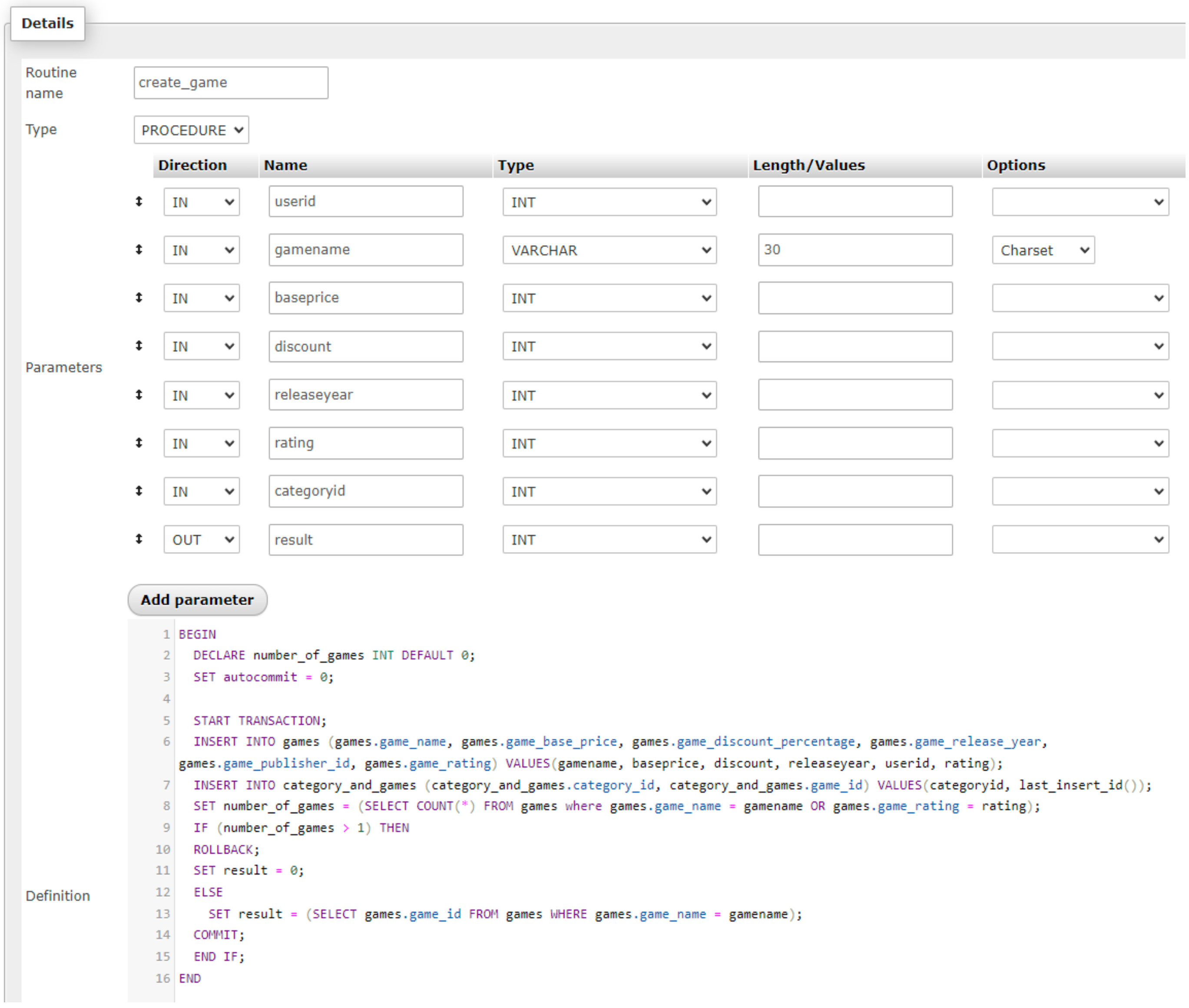
```
BEGIN
  DECLARE number_of_games INT DEFAULT 0;
  SET autocommit = 0;

  START TRANSACTION;
  INSERT INTO comments (comments.comment_game_id, comments.comment_customer_id, comments.comment_content) VALUES(gameid, userid, content);
  SET number_of_games = (SELECT COUNT(*) FROM comments WHERE comments.comment_game_id = gameid AND comments.comment_customer_id = userid);

  IF (number_of_games > 1) THEN
  ROLLBACK;
  SET result = 0;
  ELSE
    SET result = 1;
    COMMIT;
  END IF;
```

Name:

create_game

Definition:

Starts an atomic transaction. Inserts into games with taken inputs. Inserts into category_and_games with taken inputs. If there are more than 1 games with same same game_name and game_rating it rollbacks and output results becomes 0, else commits and output result becomes recently created game_id.



Routine name	create_game
Type	PROCEDURE

Parameters

Direction	Name	Type	Length/Values	Options
IN	userid	INT		
IN	gamename	VARCHAR	30	Charset
IN	baseprice	INT		
IN	discount	INT		
IN	releaseyear	INT		
IN	rating	INT		
IN	categoryid	INT		
OUT	result	INT		

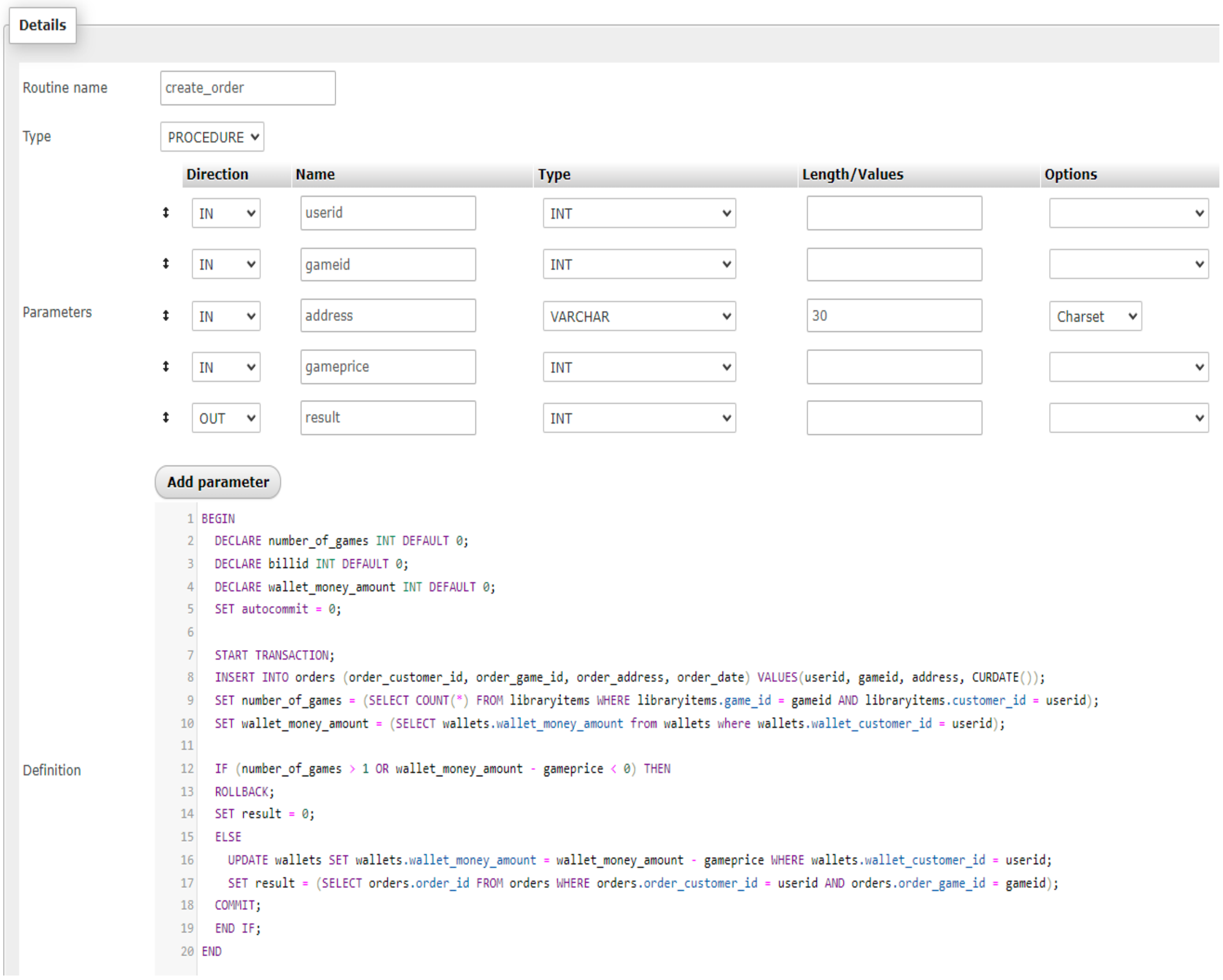
Definition

```sql
BEGIN
  DECLARE number_of_games INT DEFAULT 0;
  SET autocommit = 0;

  START TRANSACTION;
  INSERT INTO games (games.game_name, games.game_base_price, games.game_discount_percentage, games.game_release_year,
games.game_publisher_id, games.game_rating) VALUES(gamename, baseprice, discount, releaseyear, userid, rating);
  INSERT INTO category_and_games (category_and_games.category_id, category_and_games.game_id) VALUES(categoryid, last_insert_id());
  SET number_of_games = (SELECT COUNT(*) FROM games where games.game_name = gamename OR games.game_rating = rating);
  IF (number_of_games > 1) THEN
  ROLLBACK;
  SET result = 0;
  ELSE
    SET result = (SELECT games.game_id FROM games WHERE games.game_name = gamename);
  COMMIT;
  END IF;
END
```

Name:

create_order

Definition:
Starts an atomic transaction. Inserts into orders new order with taken inputs. If there are more than 1 library_items with same customer_id and game_id or wallet_money_amount is less than game_net_price or gameprice is less or equal to 0 then it rollbacks and output result is 0. Else it commits and output result is recently created order_id. It has 3 triggers: create_bill, create_library_item and create_notification which I explained on previous pages.



Routine name: create_order

Type: PROCEDURE

Parameters:

Direction	Name	Type	Length/Values	Options
IN	userid	INT		
IN	gameid	INT		
IN	address	VARCHAR	30	Charset
IN	gameprice	INT		
OUT	result	INT		

Definition:

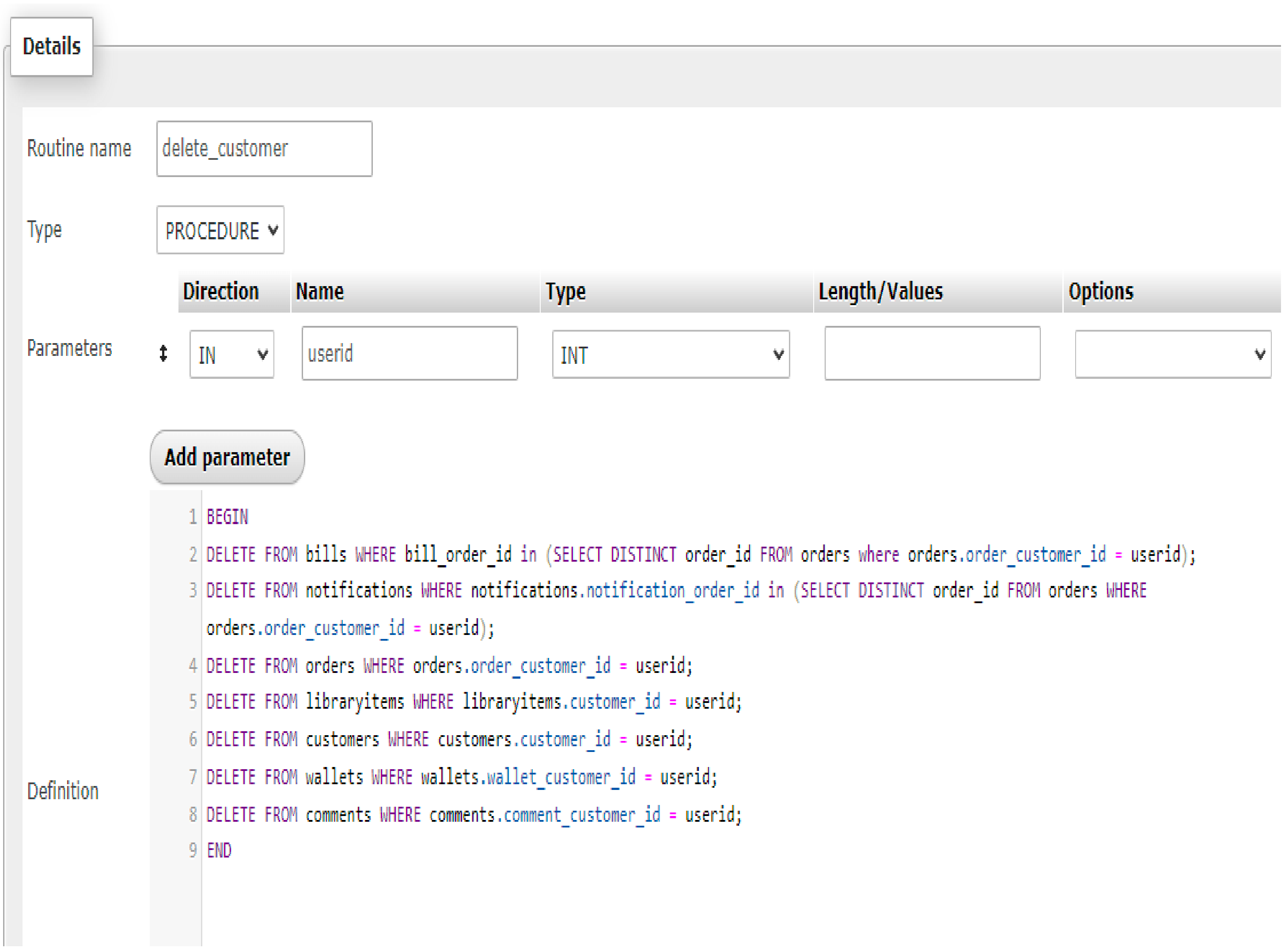
```
BEGIN
  DECLARE number_of_games INT DEFAULT 0;
  DECLARE billid INT DEFAULT 0;
  DECLARE wallet_money_amount INT DEFAULT 0;
  SET autocommit = 0;

  START TRANSACTION;
  INSERT INTO orders (order_customer_id, order_game_id, order_address, order_date) VALUES(userid, gameid, address, CURDATE());
  SET number_of_games = (SELECT COUNT(*) FROM libraryitems WHERE libraryitems.game_id = gameid AND libraryitems.customer_id = userid);
  SET wallet_money_amount = (SELECT wallets.wallet_money_amount from wallets where wallets.wallet_customer_id = userid);

  IF (number_of_games > 1 OR wallet_money_amount - gameprice < 0) THEN
  ROLLBACK;
  SET result = 0;
  ELSE
    UPDATE wallets SET wallets.wallet_money_amount = wallet_money_amount - gameprice WHERE wallets.wallet_customer_id = userid;
    SET result = (SELECT orders.order_id FROM orders WHERE orders.order_customer_id = userid AND orders.order_game_id = gameid);
  COMMIT;
  END IF;
END
```

Name:

delete_customer

Definition:

It takes userid as input. It deletes from bills, orders, library_items, comments, notifications and customers using input userid.



Routine name	delete_customer
Type	PROCEDURE

Parameters:

Direction	Name	Type	Length/Values	Options
IN	userid	INT		

Definition

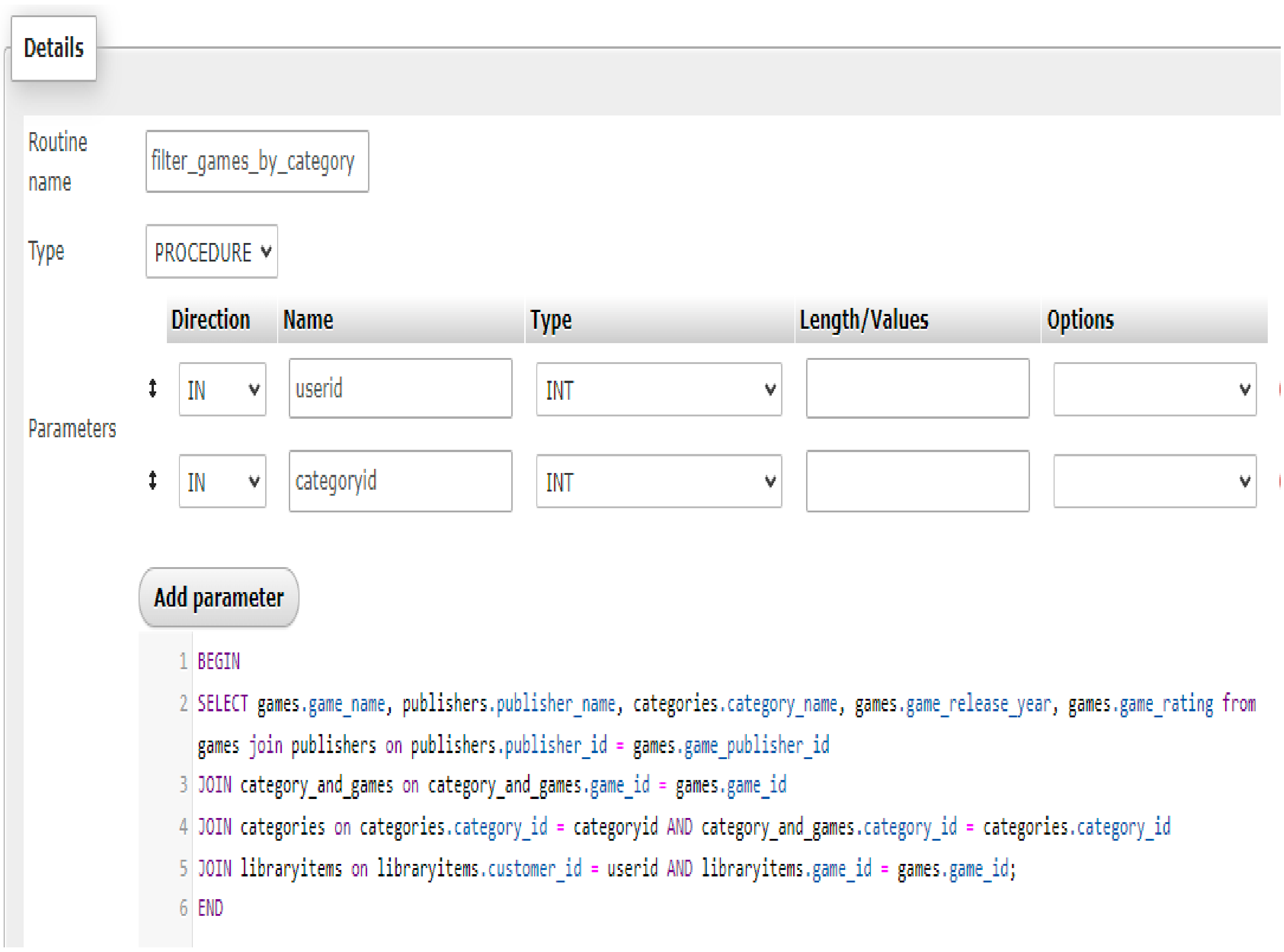
```sql
BEGIN
DELETE FROM bills WHERE bill_order_id in (SELECT DISTINCT order_id FROM orders where orders.order_customer_id = userid);
DELETE FROM notifications WHERE notifications.notification_order_id in (SELECT DISTINCT order_id FROM orders WHERE
orders.order_customer_id = userid);
DELETE FROM orders WHERE orders.order_customer_id = userid;
DELETE FROM libraryitems WHERE libraryitems.customer_id = userid;
DELETE FROM customers WHERE customers.customer_id = userid;
DELETE FROM wallets WHERE wallets.wallet_customer_id = userid;
DELETE FROM comments WHERE comments.comment_customer_id = userid;
END
```

Name:

filter_games_by_category

Definition:

It uses inner joins and returns the games that customer has that is being stored at library_items table. Uses INNER JOINs.



Routine name: filter_games_by_category

Type: PROCEDURE

Parameters

Direction	Name	Type	Length/Values	Options
IN	userid	INT		
IN	categoryid	INT		

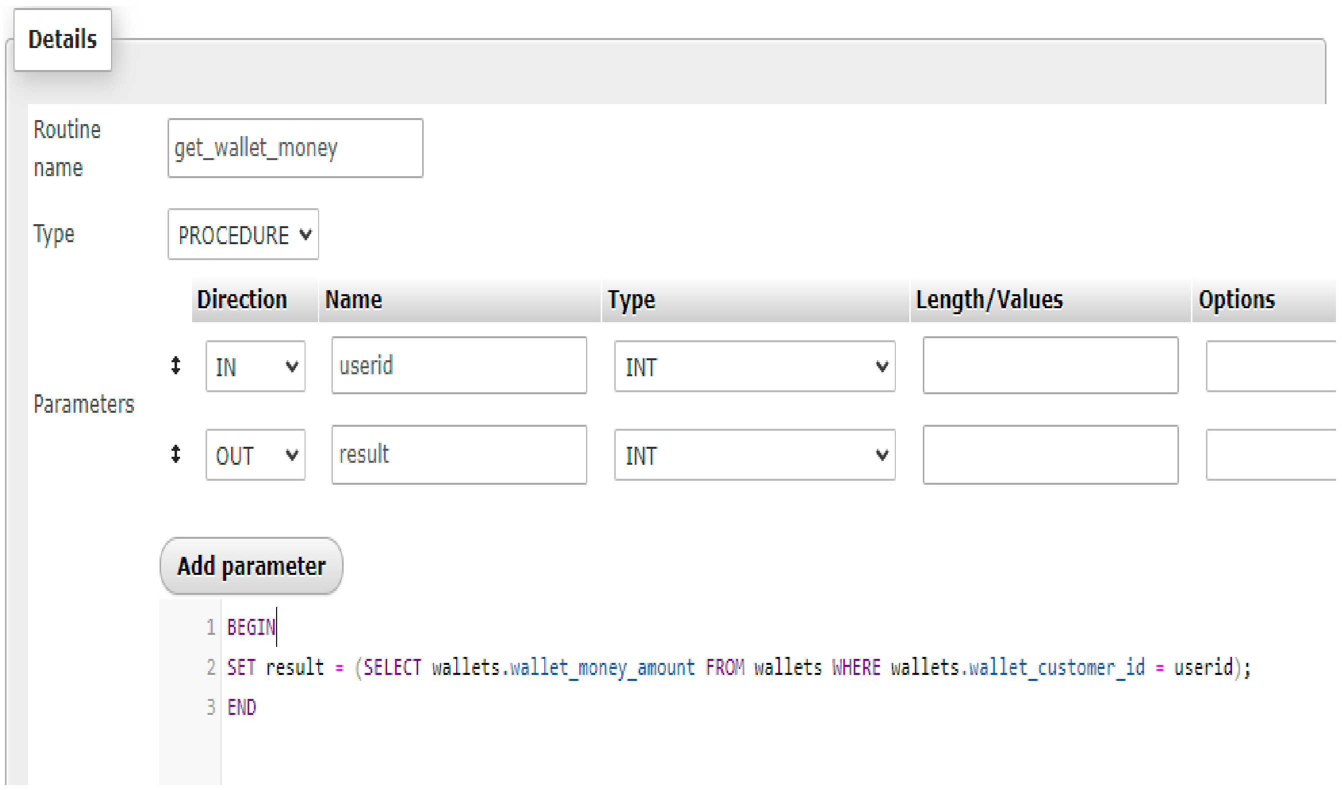
```
BEGIN
SELECT games.game_name, publishers.publisher_name, categories.category_name, games.game_release_year, games.game_rating from games join publishers on publishers.publisher_id = games.game_publisher_id
JOIN category_and_games on category_and_games.game_id = games.game_id
JOIN categories on categories.category_id = categoryid AND category_and_games.category_id = categories.category_id
JOIN libraryitems on libraryitems.customer_id = userid AND libraryitems.game_id = games.game_id;
END
```

Name:

get_wallet_money

Definition:

It takes userid as input. It returns the wallet_money_amount where wallet_customer_id = userid.



Details

Routine name	get_wallet_money
Type	PROCEDURE

Parameters

Direction	Name	Type	Length/Values	Options
IN	userid	INT		
OUT	result	INT		

Add parameter

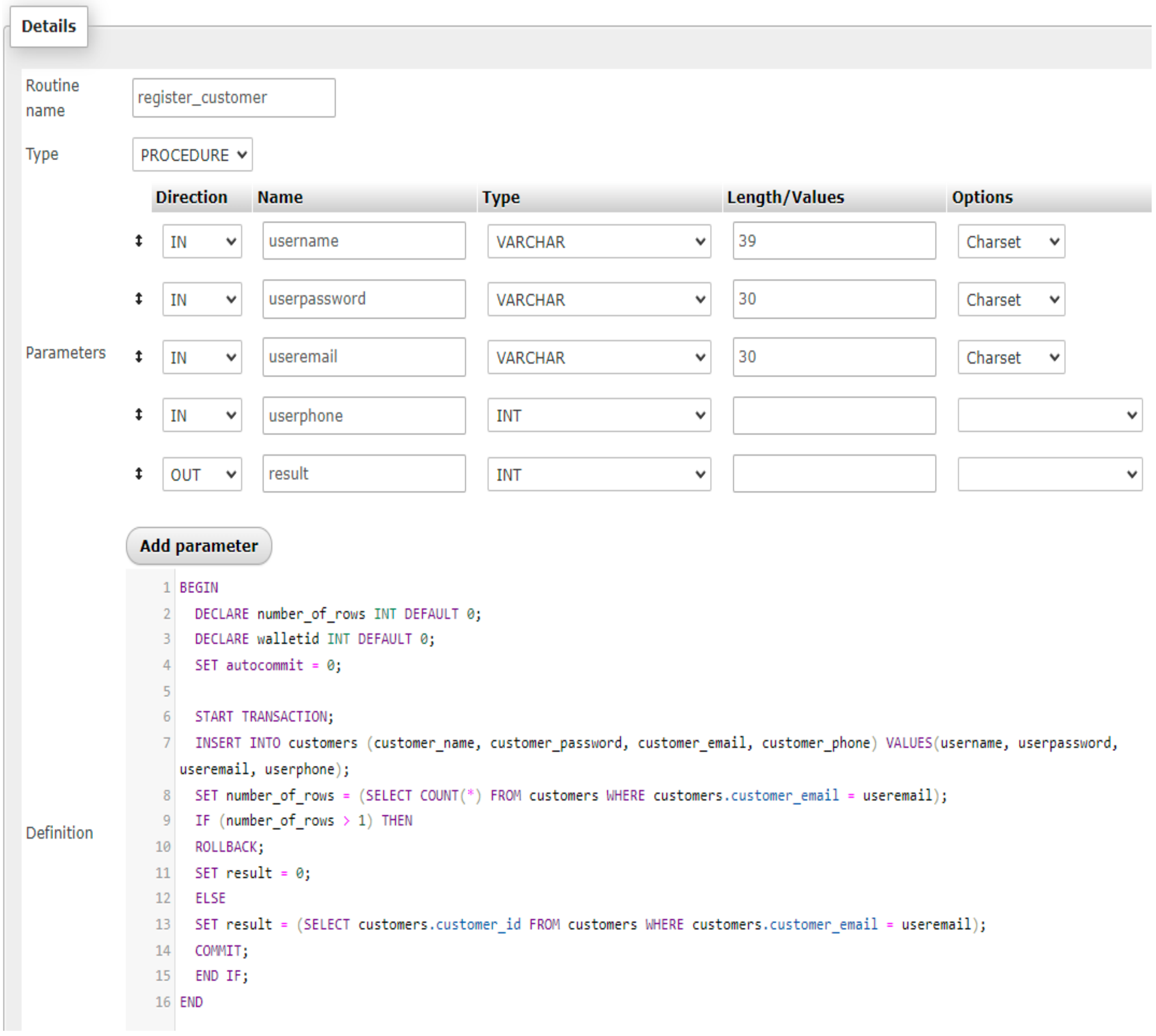
```
BEGIN
SET result = (SELECT wallets.wallet_money_amount FROM wallets WHERE wallets.wallet_customer_id = userid);
END
```

Name:

register_customer

Definition:

Starts an atomic transaction. Inserts into customers new customer with taken inputs. If there are more than 1 customers with same email then it rollbacks and output result is 0. Else it commits and output result is recently created customer_id.



Routine name	register_customer
Type	PROCEDURE

Parameters

Direction	Name	Type	Length/Values	Options
IN	username	VARCHAR	39	Charset
IN	userpassword	VARCHAR	30	Charset
IN	useremail	VARCHAR	30	Charset
IN	userphone	INT		
OUT	result	INT		

Definition

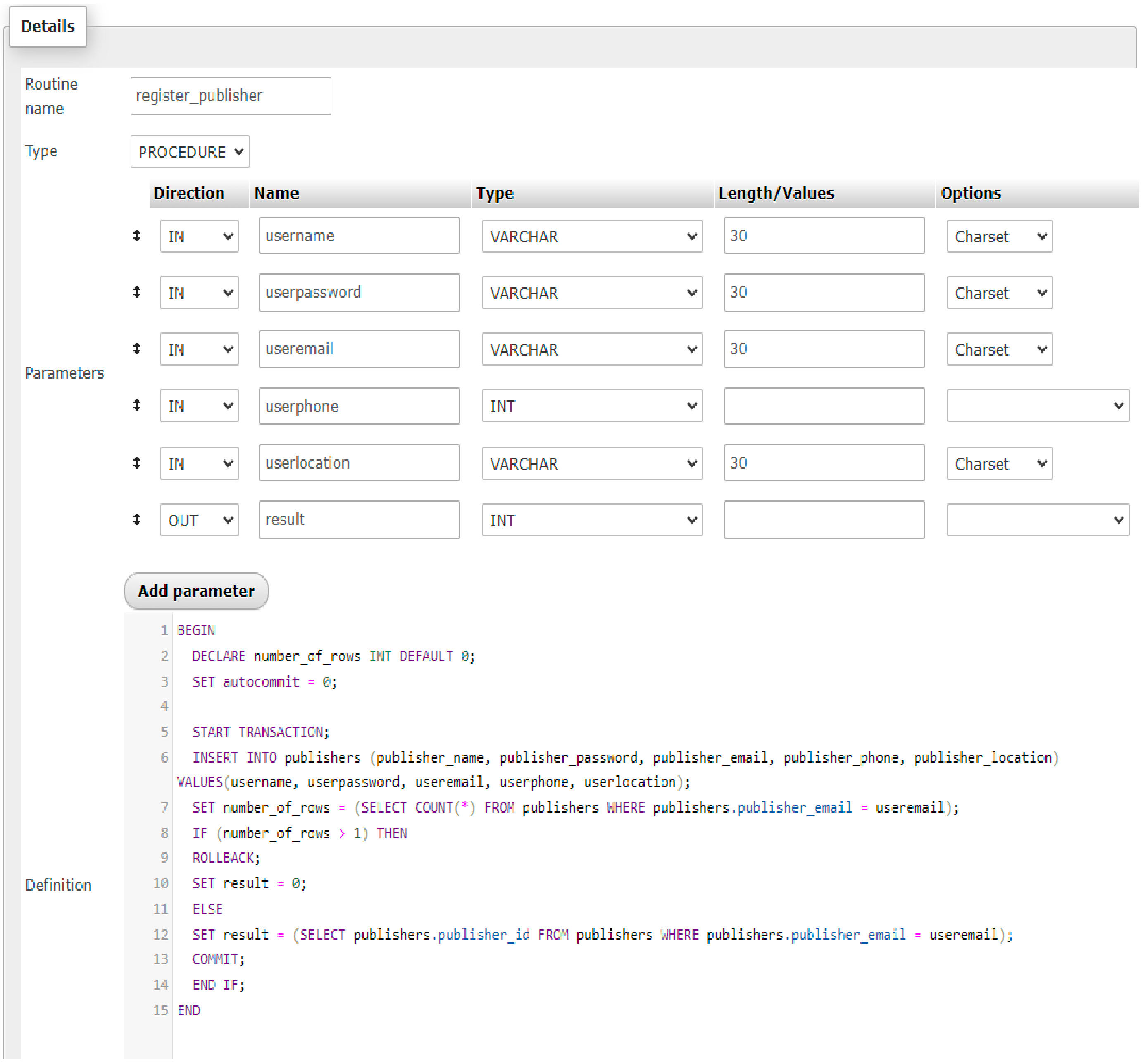
```
BEGIN
  DECLARE number_of_rows INT DEFAULT 0;
  DECLARE walletid INT DEFAULT 0;
  SET autocommit = 0;

  START TRANSACTION;
  INSERT INTO customers (customer_name, customer_password, customer_email, customer_phone) VALUES(username, userpassword,
useremail, userphone);
  SET number_of_rows = (SELECT COUNT(*) FROM customers WHERE customers.customer_email = useremail);
  IF (number_of_rows > 1) THEN
  ROLLBACK;
  SET result = 0;
  ELSE
  SET result = (SELECT customers.customer_id FROM customers WHERE customers.customer_email = useremail);
  COMMIT;
  END IF;
END
```

Name:

register_publisher

Definition:

Starts an atomic transaction. Inserts into publishers new publisher with taken inputs. If there are more than 1 publishers with same email then it rollbacks and output result is 0. Else it commits and output result is recently created publisher_id.



Routine name: register_publisher

Type: PROCEDURE

Parameters

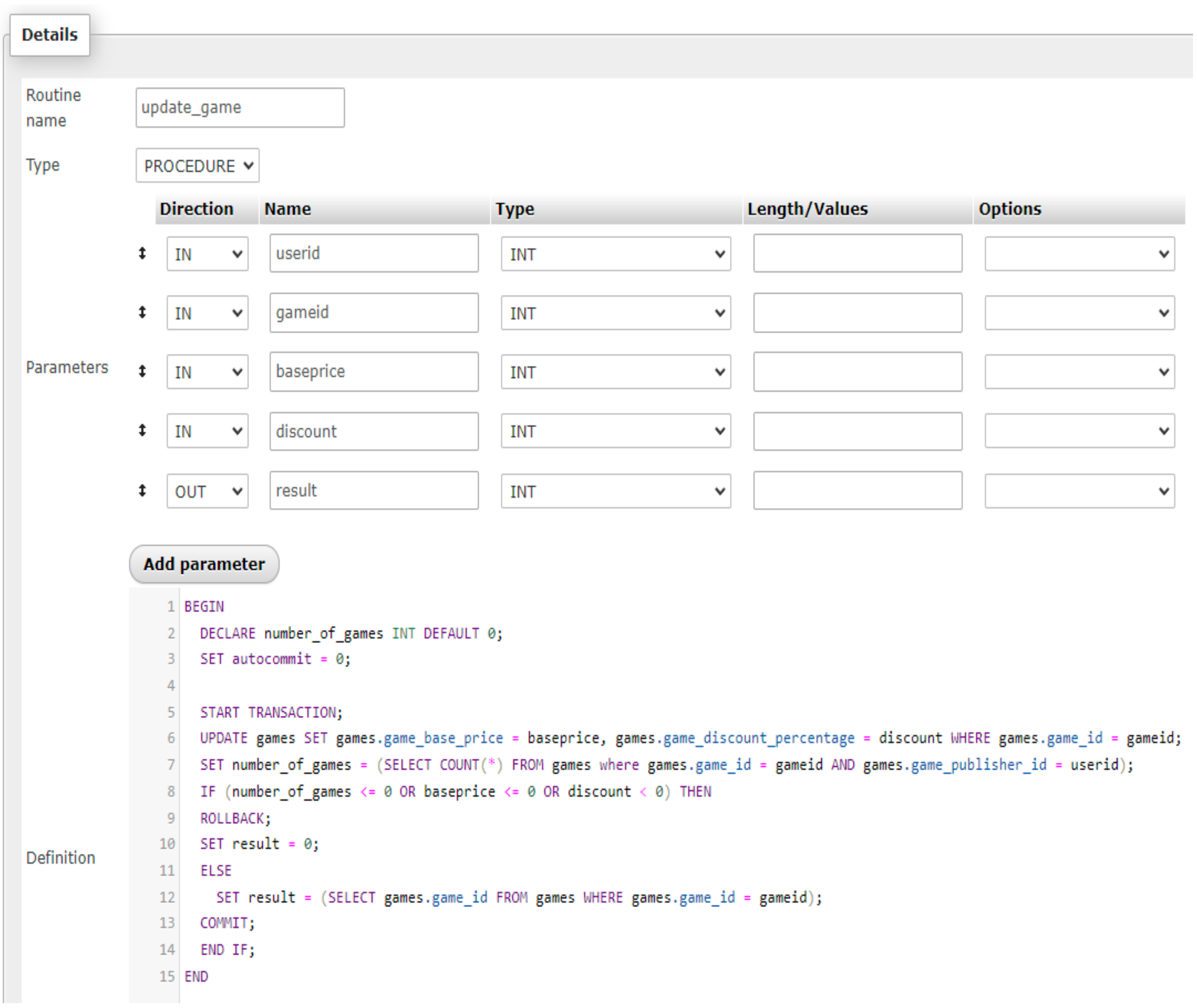
Direction	Name	Type	Length/Values	Options
IN	username	VARCHAR	30	Charset
IN	userpassword	VARCHAR	30	Charset
IN	useremail	VARCHAR	30	Charset
IN	userphone	INT		
IN	userlocation	VARCHAR	30	Charset
OUT	result	INT		

Definition

```
BEGIN
  DECLARE number_of_rows INT DEFAULT 0;
  SET autocommit = 0;

  START TRANSACTION;
  INSERT INTO publishers (publisher_name, publisher_password, publisher_email, publisher_phone, publisher_location)
VALUES(username, userpassword, useremail, userphone, userlocation);
  SET number_of_rows = (SELECT COUNT(*) FROM publishers WHERE publishers.publisher_email = useremail);
  IF (number_of_rows > 1) THEN
  ROLLBACK;
  SET result = 0;
  ELSE
  SET result = (SELECT publishers.publisher_id FROM publishers WHERE publishers.publisher_email = useremail);
  COMMIT;
  END IF;
END
```

Name:
update_game

Definition:
Starts an atomic transaction. Updates the game's base_price and discount_percentage. Trigger sets game's net_price. If there are less than or equal to 0 publishers with same gameid and publisher_id or baseprice is less than or equal to 0 or discount is less than 0 rollbacks and sets result to 0. Else, commits and result is set to gameid. It has a trigger named update_price_update.



Routine name: update_game

Type: PROCEDURE

Parameters:

Direction	Name	Type	Length/Values	Options
IN	userid	INT		
IN	gameid	INT		
IN	baseprice	INT		
IN	discount	INT		
OUT	result	INT		

Definition:

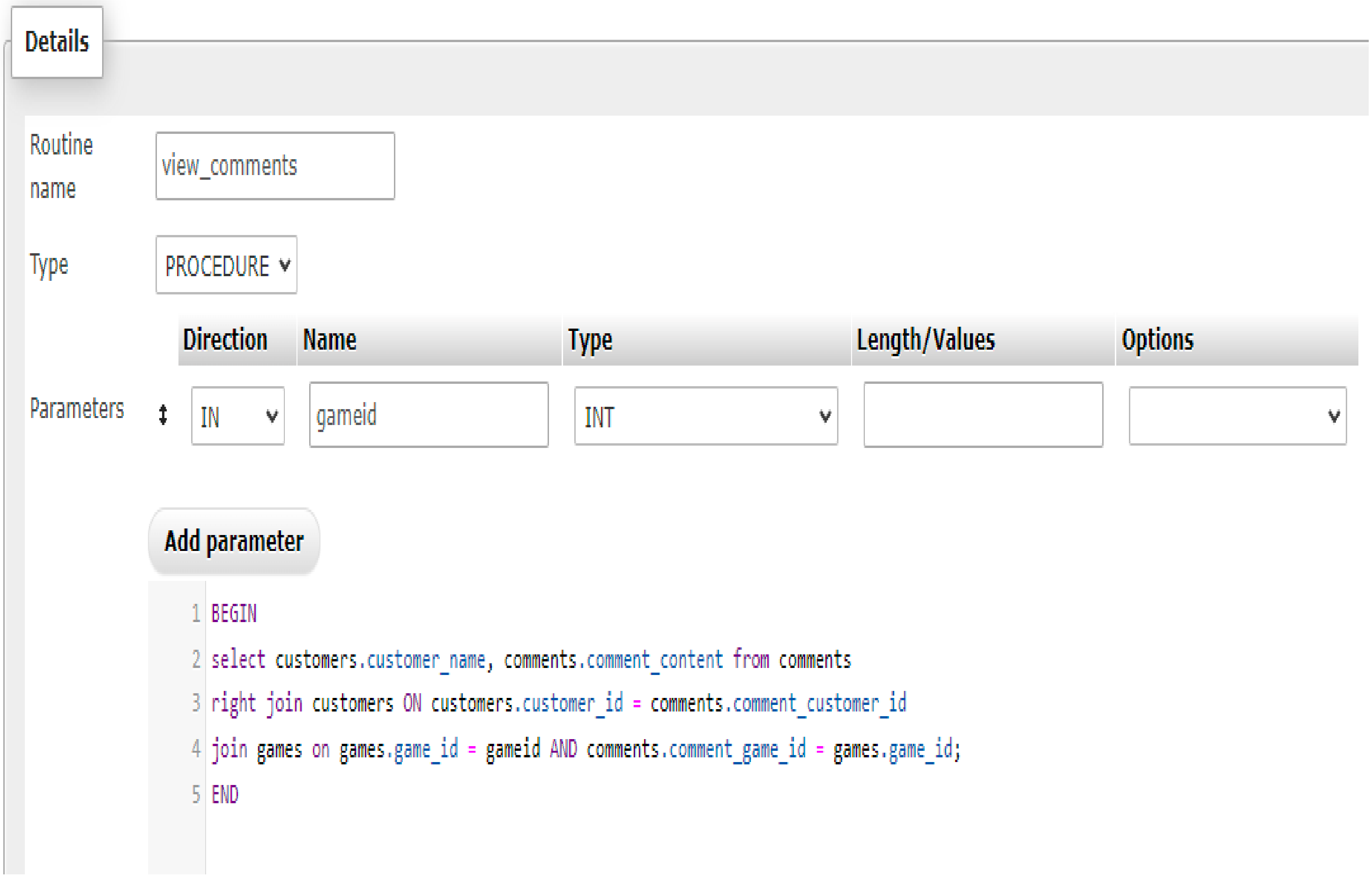
```
BEGIN
  DECLARE number_of_games INT DEFAULT 0;
  SET autocommit = 0;

  START TRANSACTION;
  UPDATE games SET games.game_base_price = baseprice, games.game_discount_percentage = discount WHERE games.game_id = gameid;
  SET number_of_games = (SELECT COUNT(*) FROM games where games.game_id = gameid AND games.game_publisher_id = userid);
  IF (number_of_games <= 0 OR baseprice <= 0 OR discount < 0) THEN
  ROLLBACK;
  SET result = 0;
  ELSE
    SET result = (SELECT games.game_id FROM games WHERE games.game_id = gameid);
  COMMIT;
  END IF;
END
```

Name:

view_comments

Definition:

Takes gameid as the input. Returns table with customer_name, comment_content WHERE customer_id = comment_customer_id, game_id = gameid. Uses RIGHT JOIN.



Routine name	view_comments
Type	PROCEDURE

Direction	Name	Type	Length/Values	Options
IN	gameid	INT		

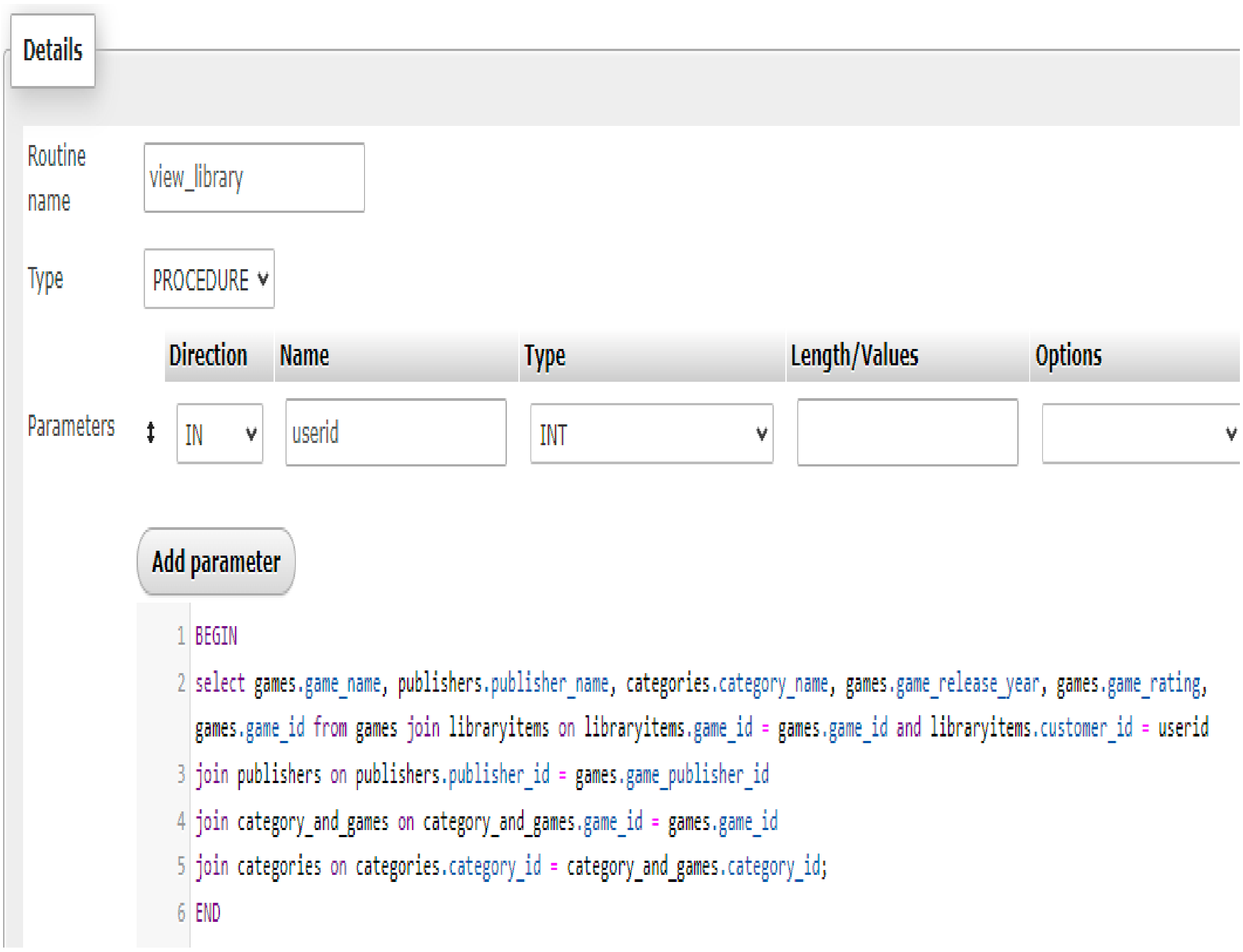
```
BEGIN
select customers.customer_name, comments.comment_content from comments
right join customers ON customers.customer_id = comments.comment_customer_id
join games on games.game_id = gameid AND comments.comment_game_id = games.game_id;
END
```

Name:

view_library

Definition:

Takes userid as input. Returns table with game_name, publisher_name, category_name, game_release_year, game_rating, game_id. This procedure allows customer to view his/her games at his/her library. Only customer can call this at user interface.



Routine name	view_library
Type	PROCEDURE

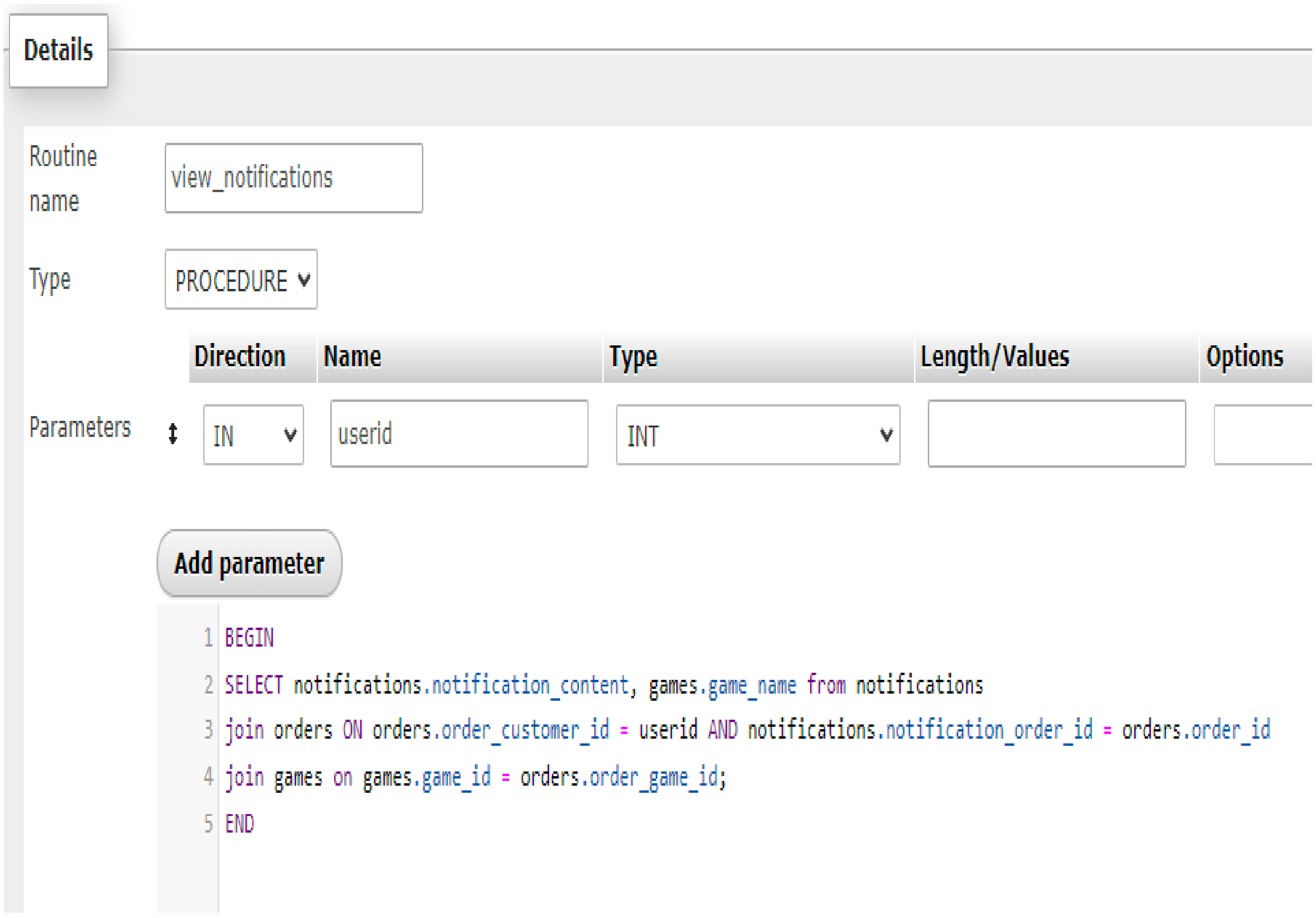
Direction	Name	Type	Length/Values	Options
IN	userid	INT		

```
BEGIN
select games.game_name, publishers.publisher_name, categories.category_name, games.game_release_year, games.game_rating,
games.game_id from games join libraryitems on libraryitems.game_id = games.game_id and libraryitems.customer_id = userid
join publishers on publishers.publisher_id = games.game_publisher_id
join category_and_games on category_and_games.game_id = games.game_id
join categories on categories.category_id = category_and_games.category_id;
END
```

Name:

view_notifications

Definition:
Takes userid as input. Returns table with notification_content, game_name from notification WHERE order_customer_id = userid AND notification_order_id = order_id AND game_id = order_game_id.



Routine name	view_notifications
Type	PROCEDURE

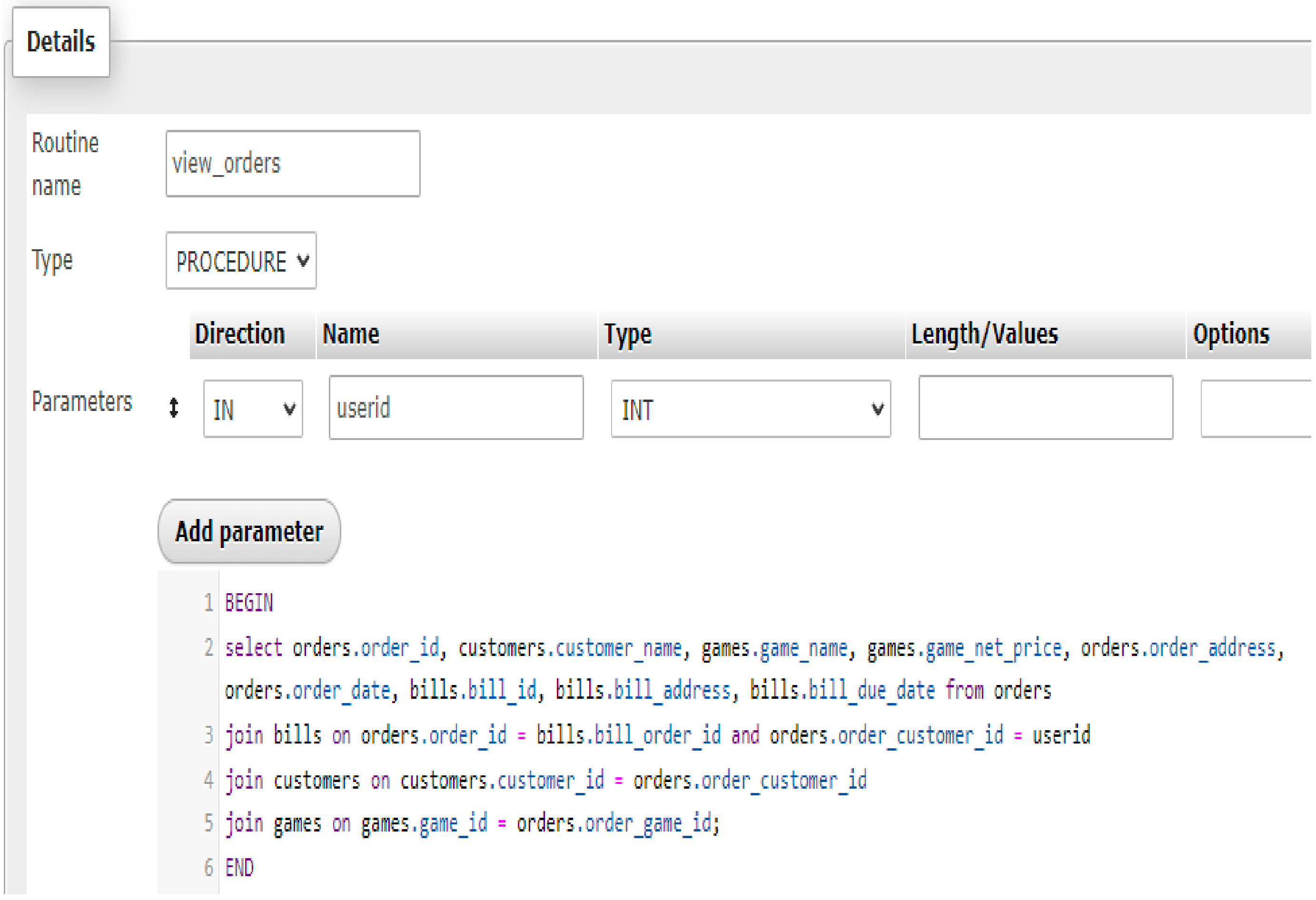
Direction	Name	Type	Length/Values	Options
IN	userid	INT		

```
BEGIN
SELECT notifications.notification_content, games.game_name from notifications
join orders ON orders.order_customer_id = userid AND notifications.notification_order_id = orders.order_id
join games on games.game_id = orders.order_game_id;
END
```

Name:

view_orders

Definition:
Takes userid as input. Returns table with order_id, game_name, game_net_price, order_address, bill_due_date
WHERE userid = order_customer_id. Uses INNER JOINs.

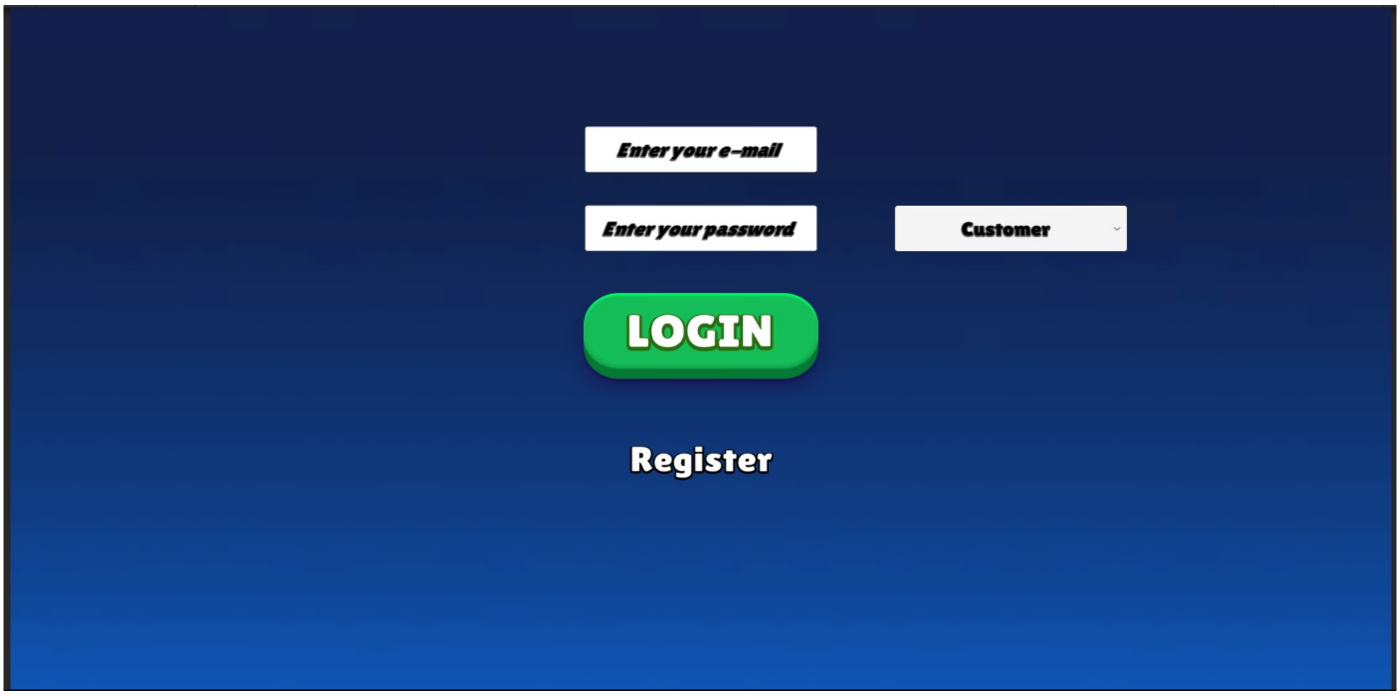


Routine name	view_orders				
Type	PROCEDURE				
	Direction	Name	Type	Length/Values	Options
Parameters	IN	userid	INT		

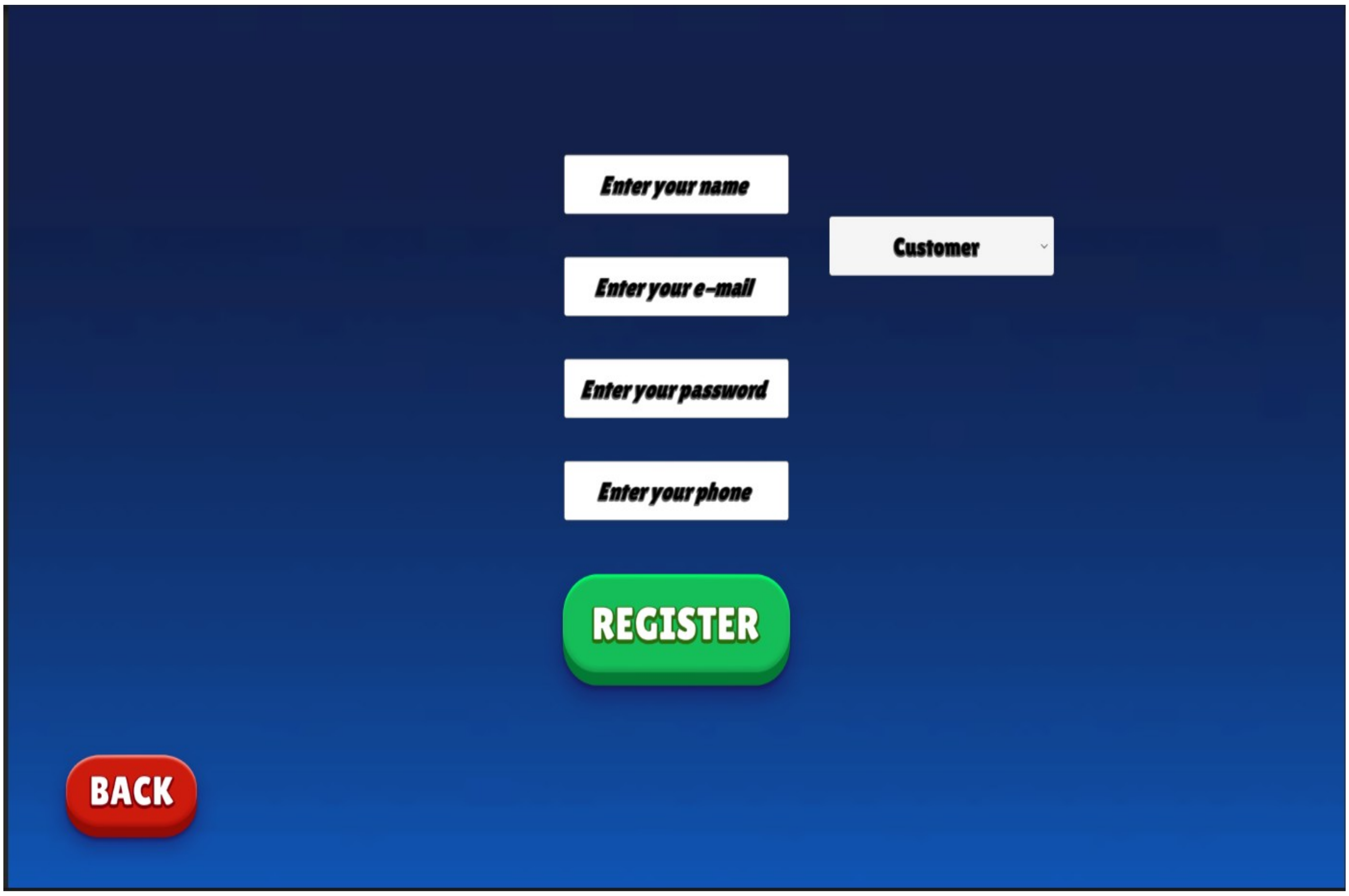
```
BEGIN
select orders.order_id, customers.customer_name, games.game_name, games.game_net_price, orders.order_address,
orders.order_date, bills.bill_id, bills.bill_address, bills.bill_due_date from orders
join bills on orders.order_id = bills.bill_order_id and orders.order_customer_id = userid
join customers on customers.customer_id = orders.order_customer_id
join games on games.game_id = orders.order_game_id;
END
```

User Interface:

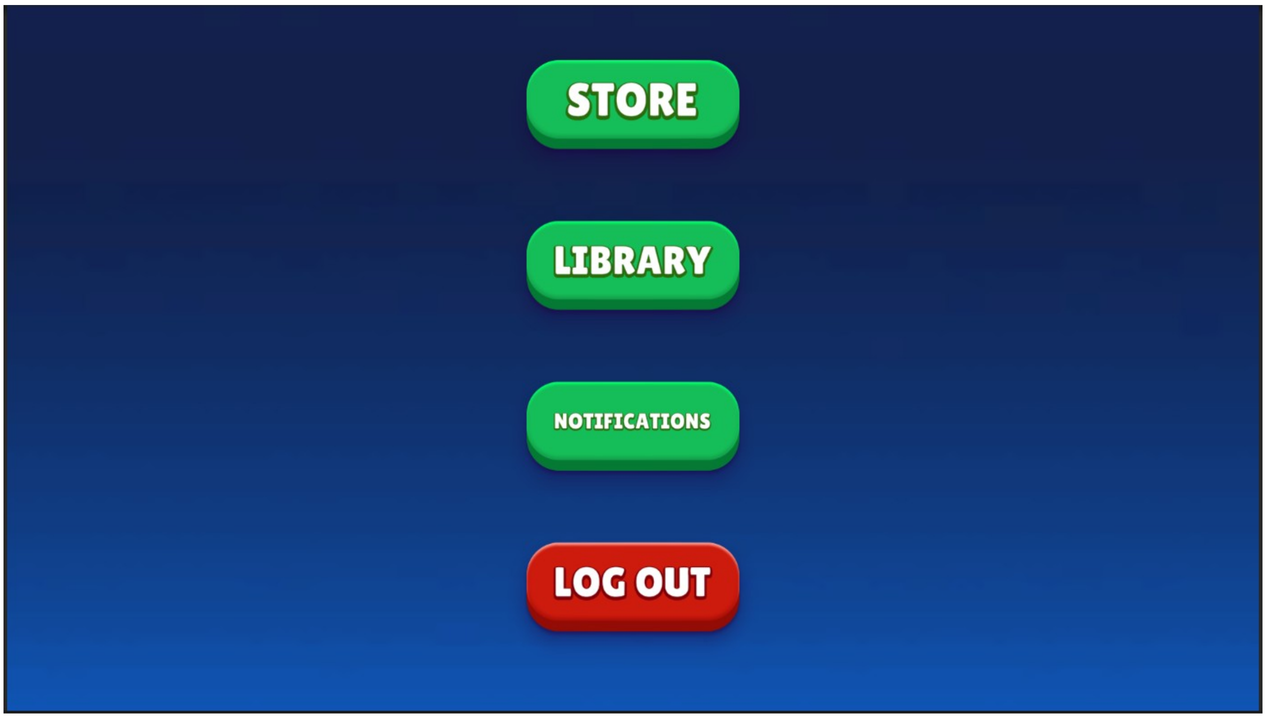
Login Panel:



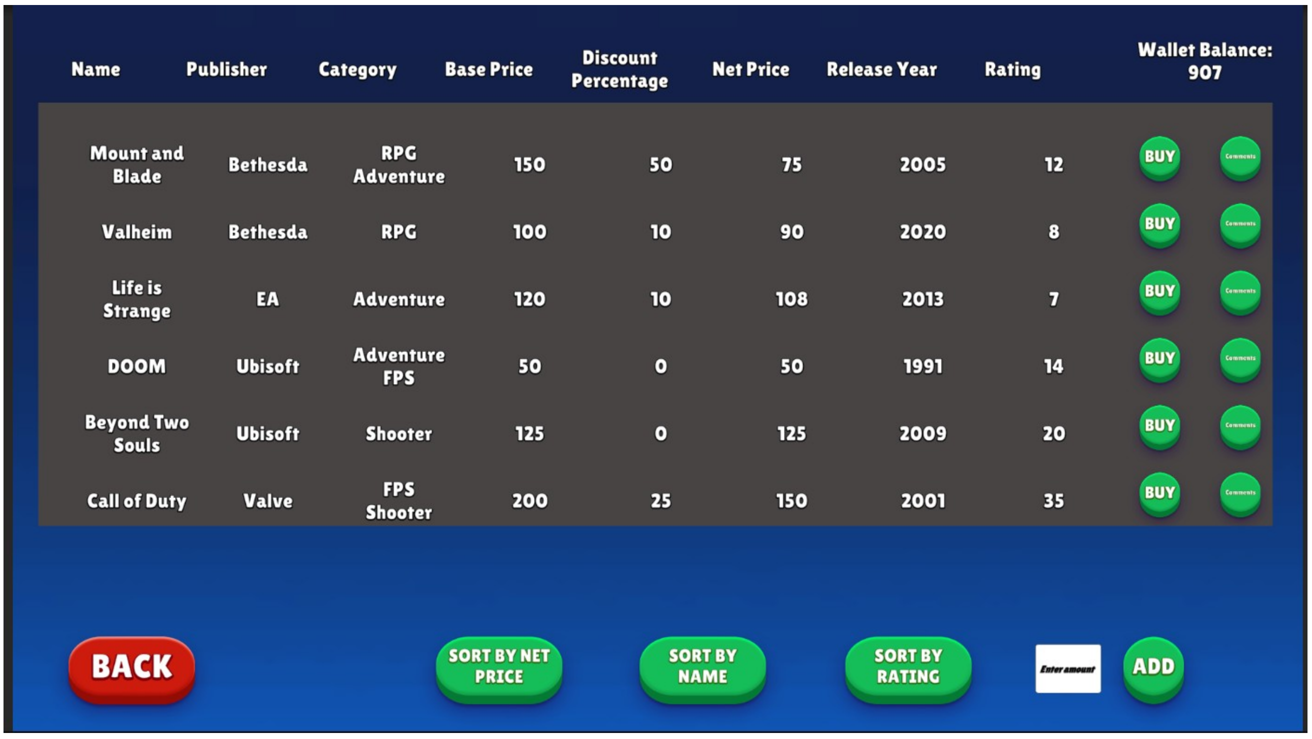
Register Panel:



Customer Main Panel:

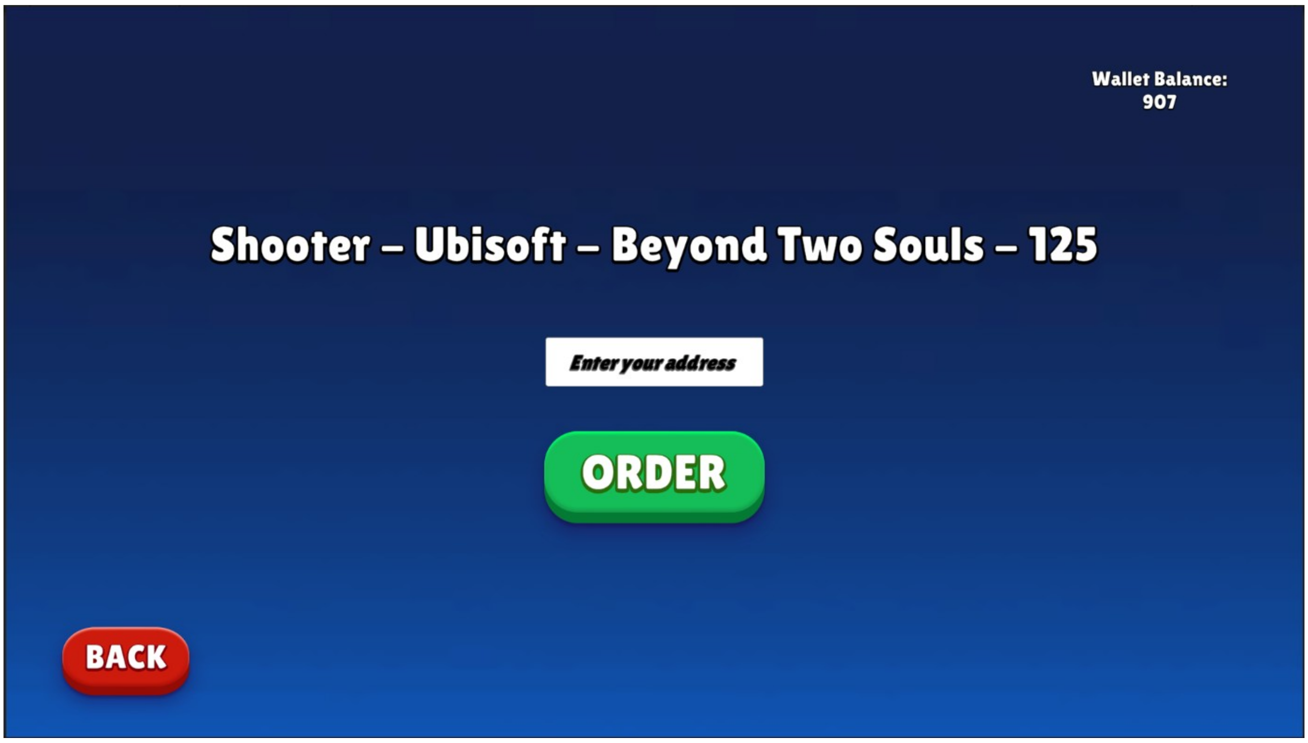


Customer Store Panel:

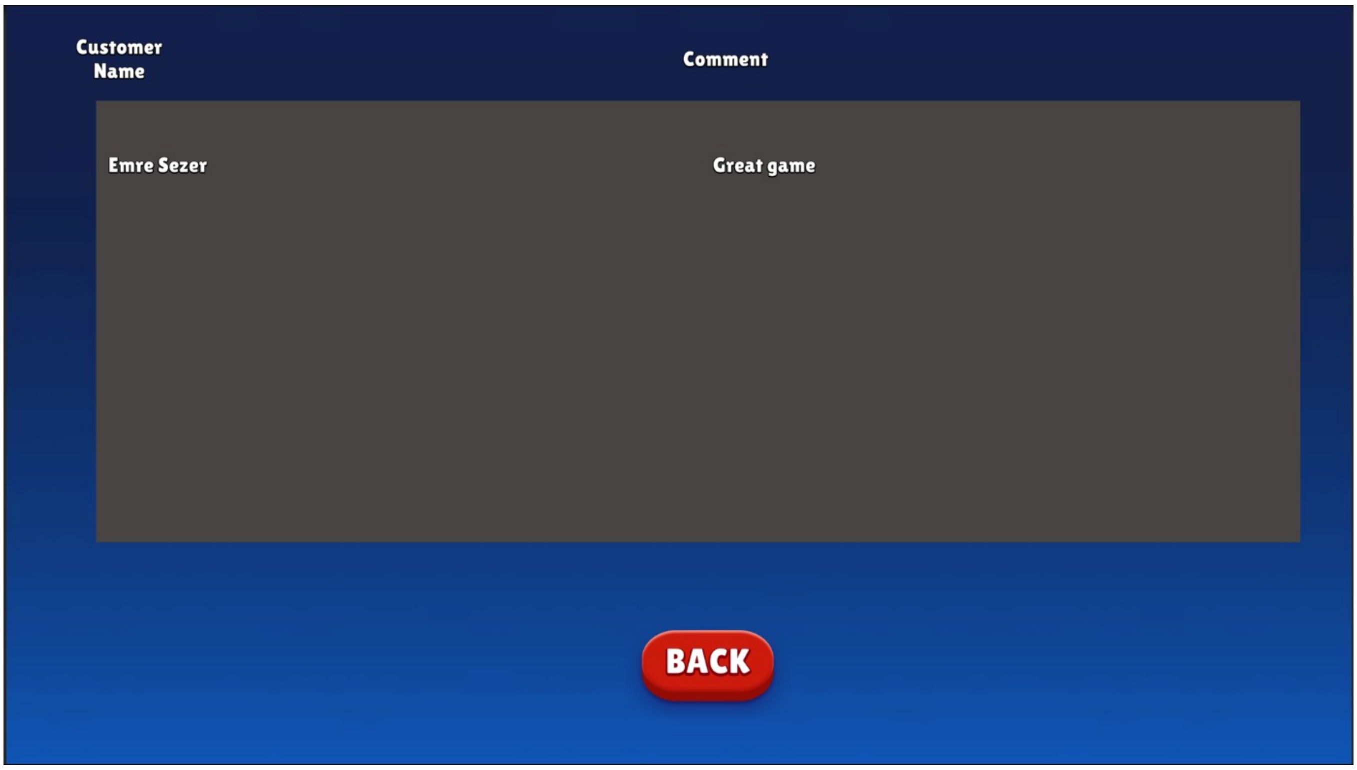


Name	Publisher	Category	Base Price	Discount Percentage	Net Price	Release Year	Rating	Wallet Balance: 907	
Mount and Blade	Bethesda	RPG Adventure	150	50	75	2005	12	BUY	Comments
Valheim	Bethesda	RPG	100	10	90	2020	8	BUY	Comments
Life is Strange	EA	Adventure	120	10	108	2013	7	BUY	Comments
DOOM	Ubisoft	Adventure FPS	50	0	50	1991	14	BUY	Comments
Beyond Two Souls	Ubisoft	Shooter	125	0	125	2009	20	BUY	Comments
Call of Duty	Valve	FPS Shooter	200	25	150	2001	35	BUY	Comments

Customer Order Panel:

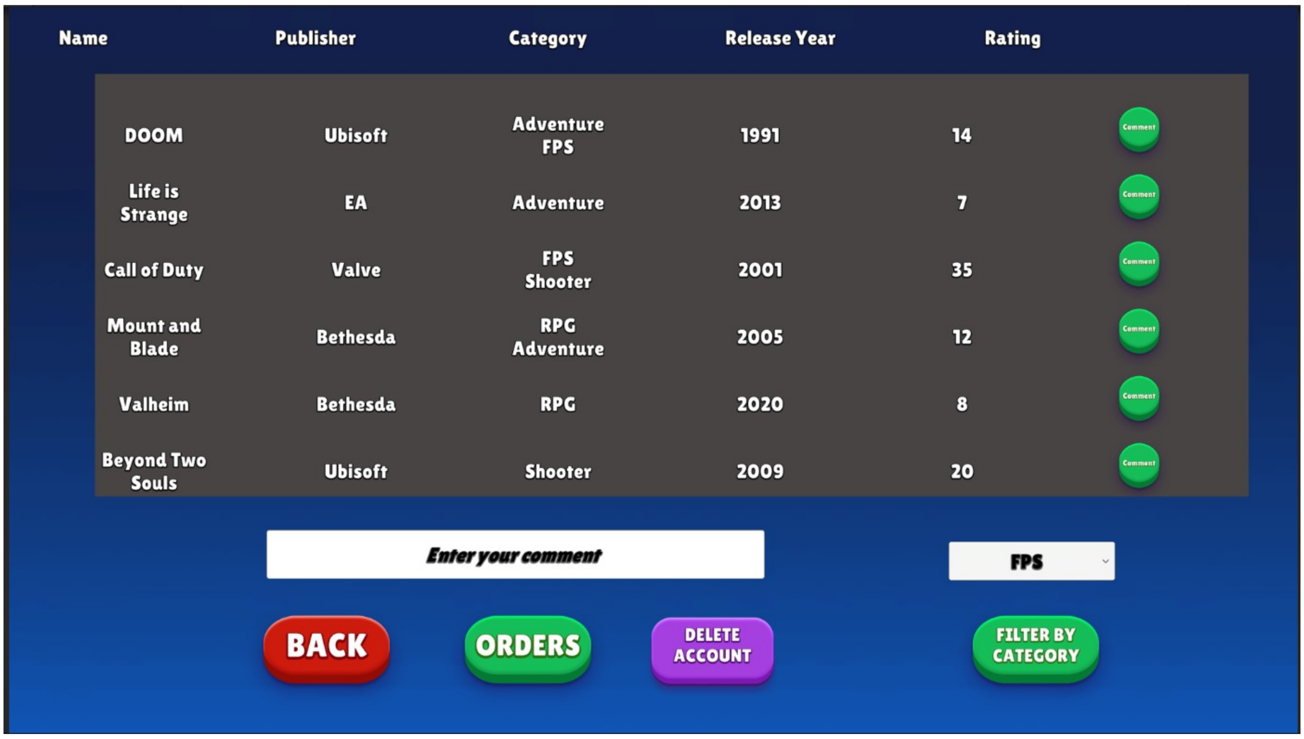


Customer View Comments Panel:



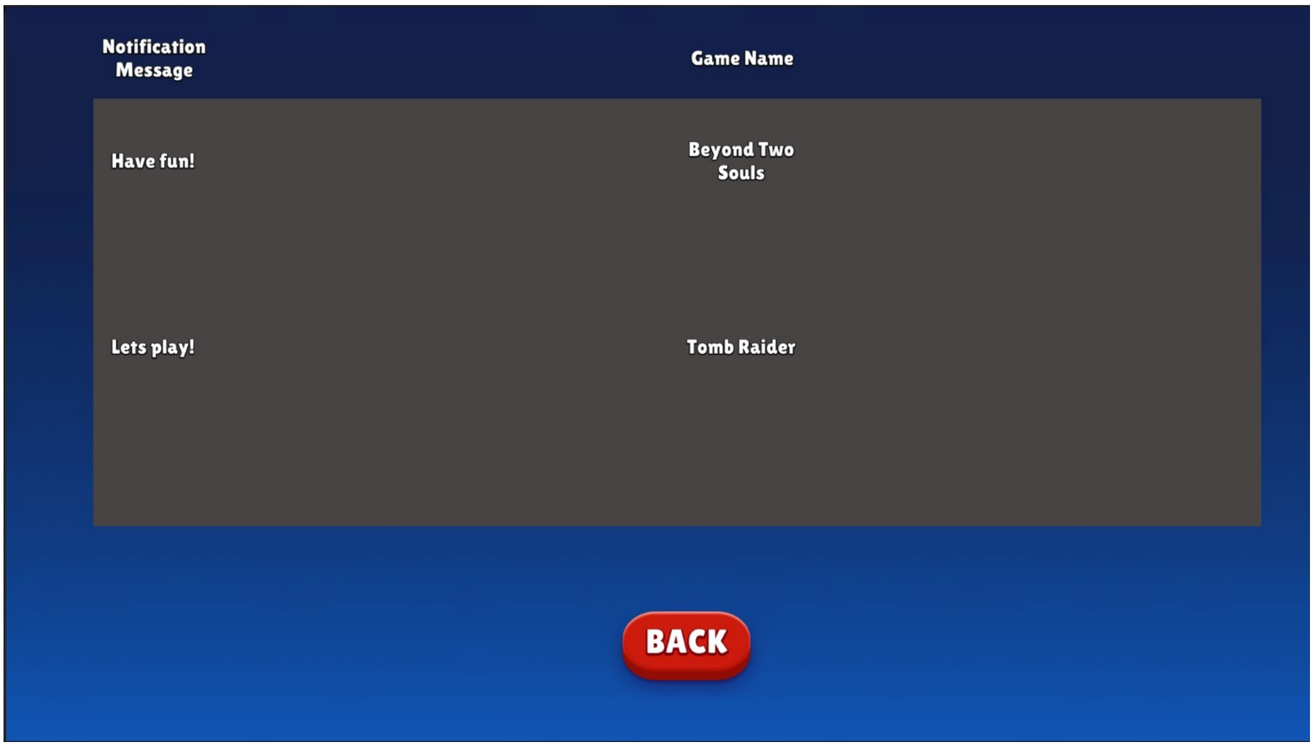
Customer Name	Comment
Emre Sezer	Great game

Customer Library Panel:



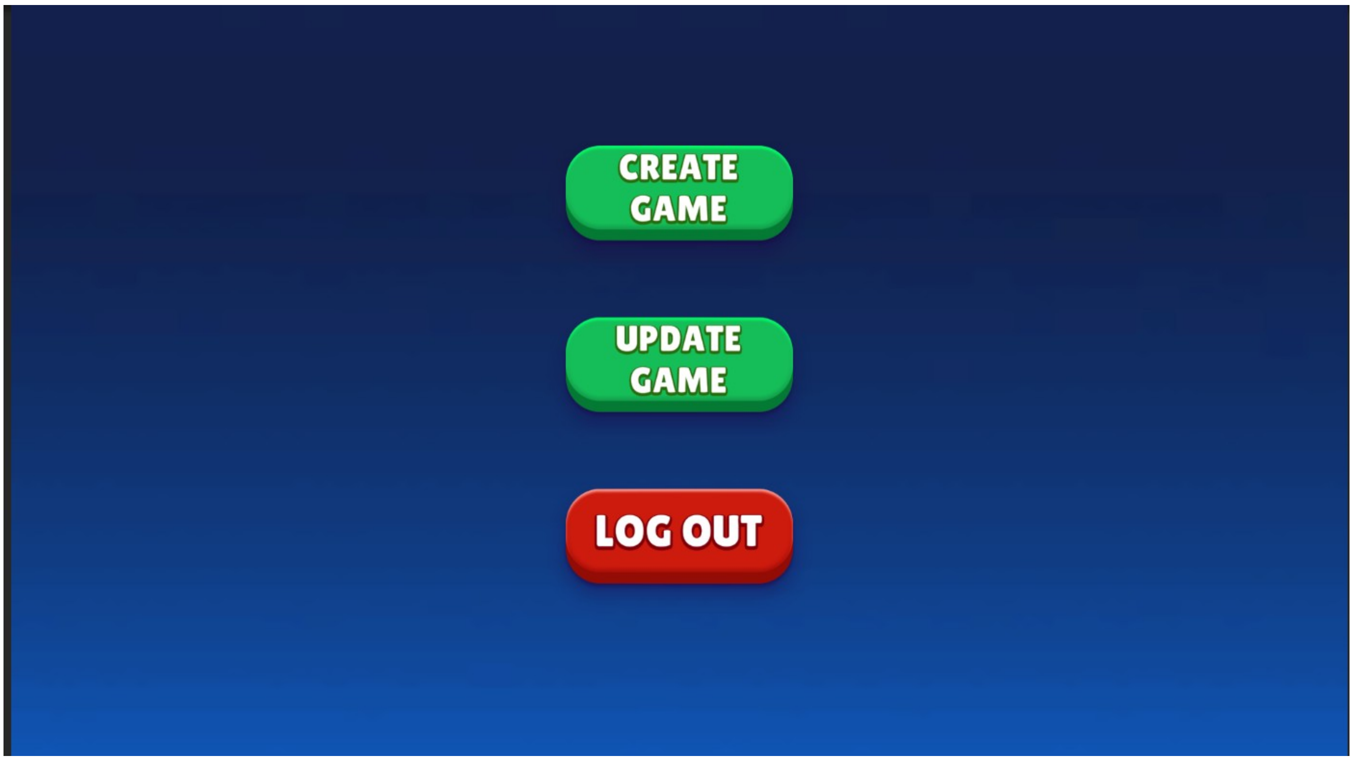
Name	Publisher	Category	Release Year	Rating	
DOOM	Ubisoft	Adventure FPS	1991	14	Comment
Life is Strange	EA	Adventure	2013	7	Comment
Call of Duty	Valve	FPS Shooter	2001	35	Comment
Mount and Blade	Bethesda	RPG Adventure	2005	12	Comment
Valheim	Bethesda	RPG	2020	8	Comment
Beyond Two Souls	Ubisoft	Shooter	2009	20	Comment

Customer Notifications Panel:

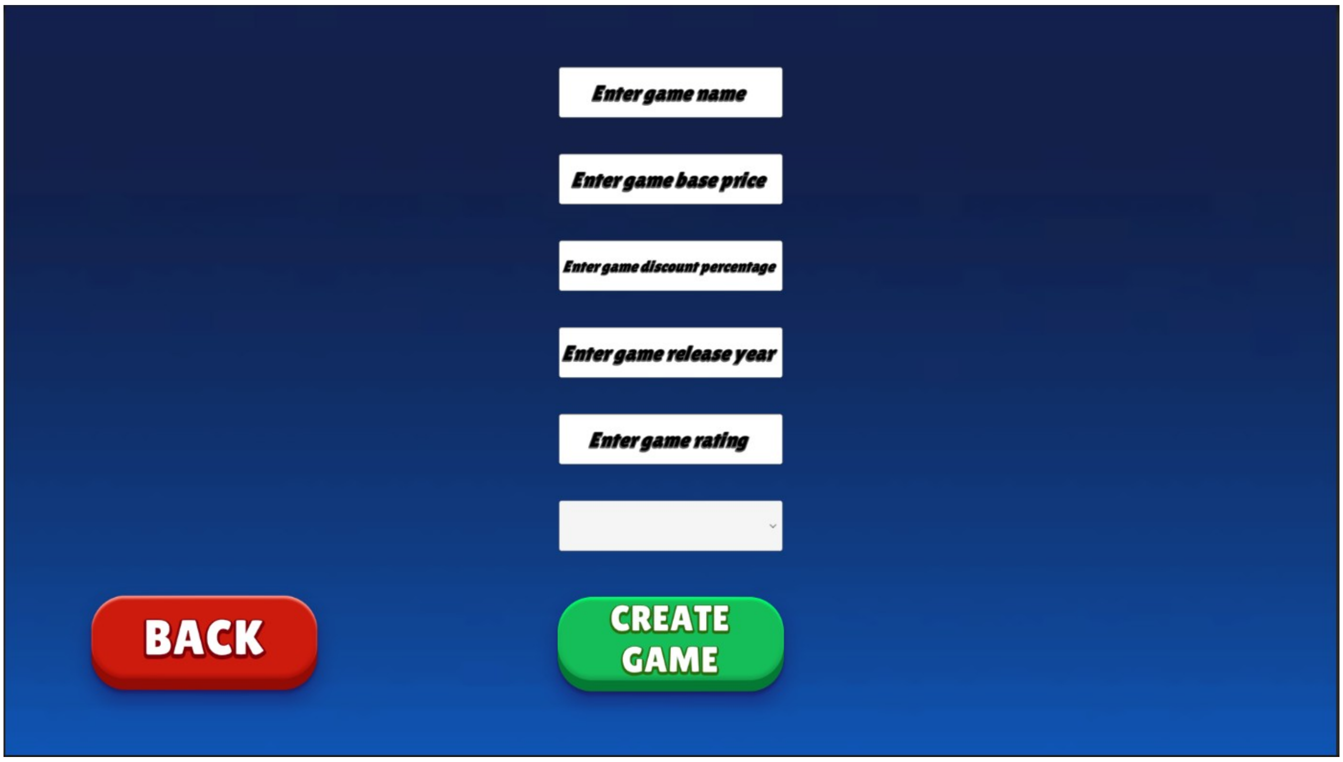


Notification Message	Game Name
Have fun!	Beyond Two Souls
Lets play!	Tomb Raider

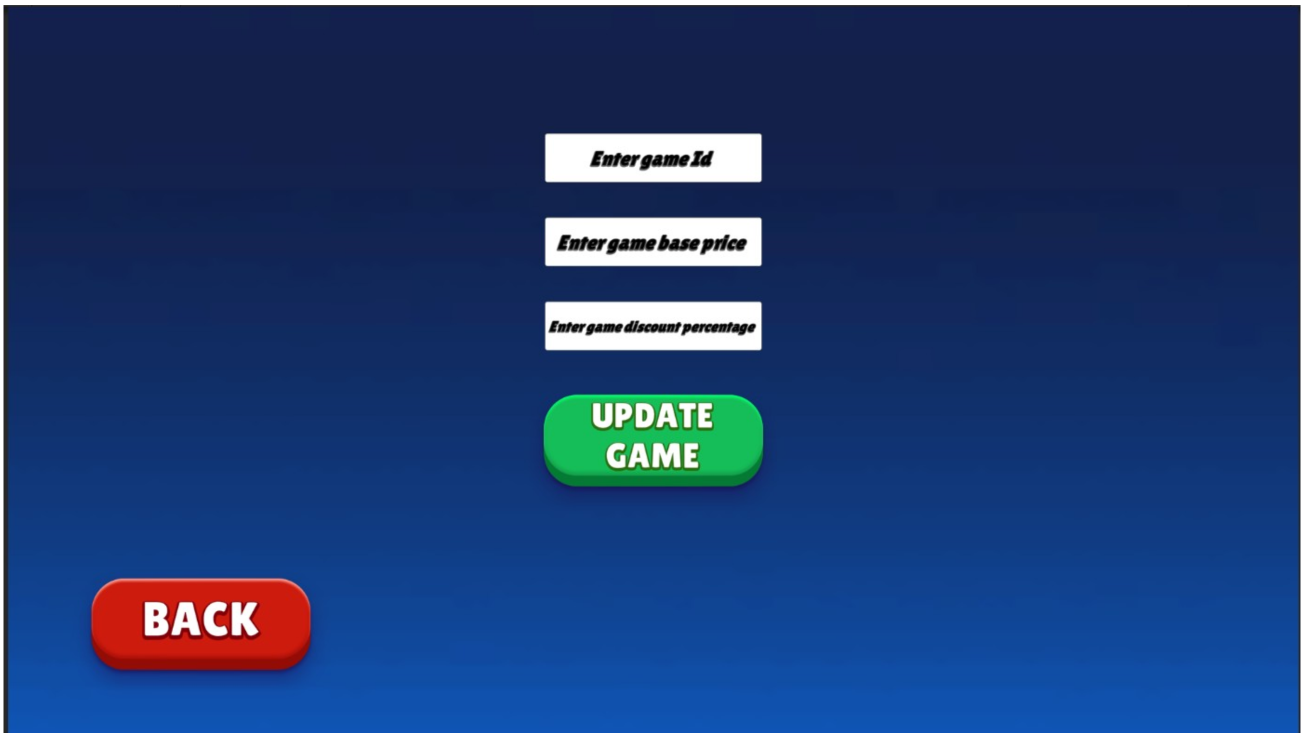
Publisher Main Panel:



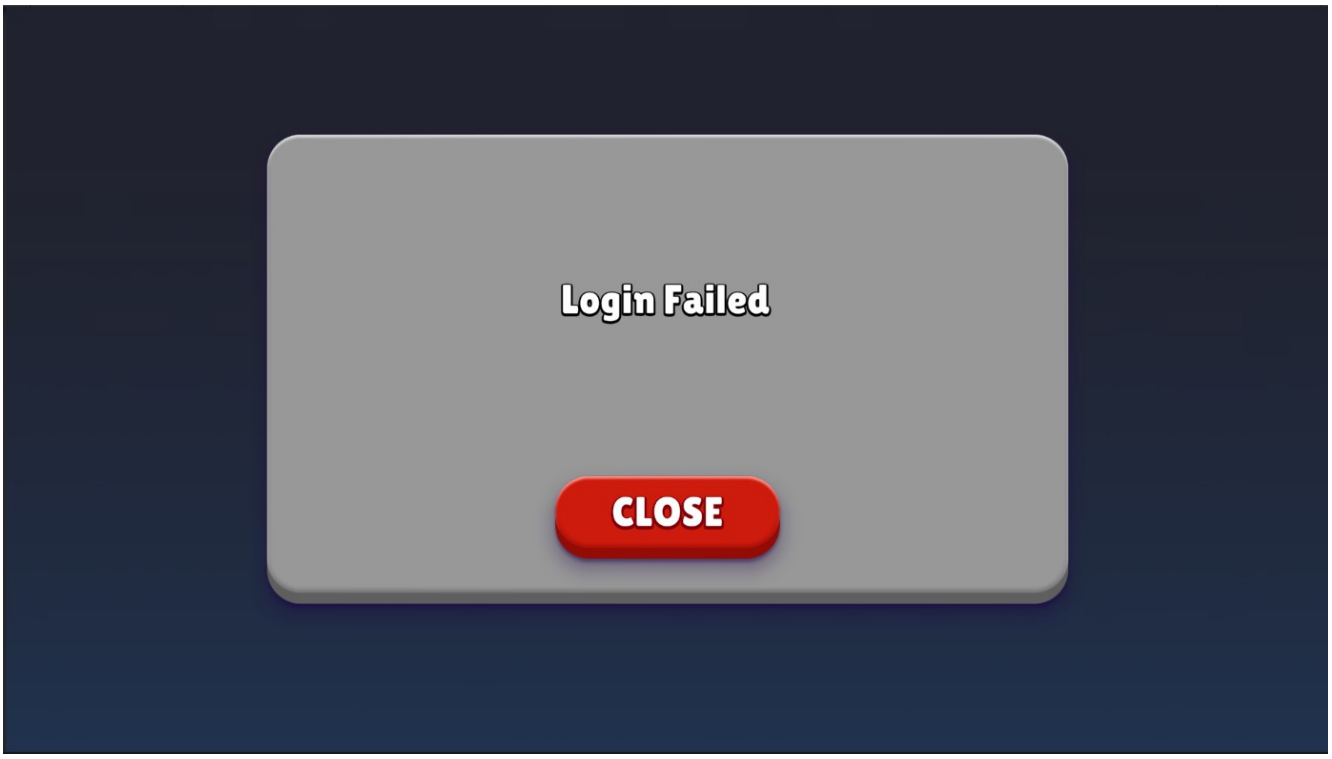
Publisher Create Game Panel:



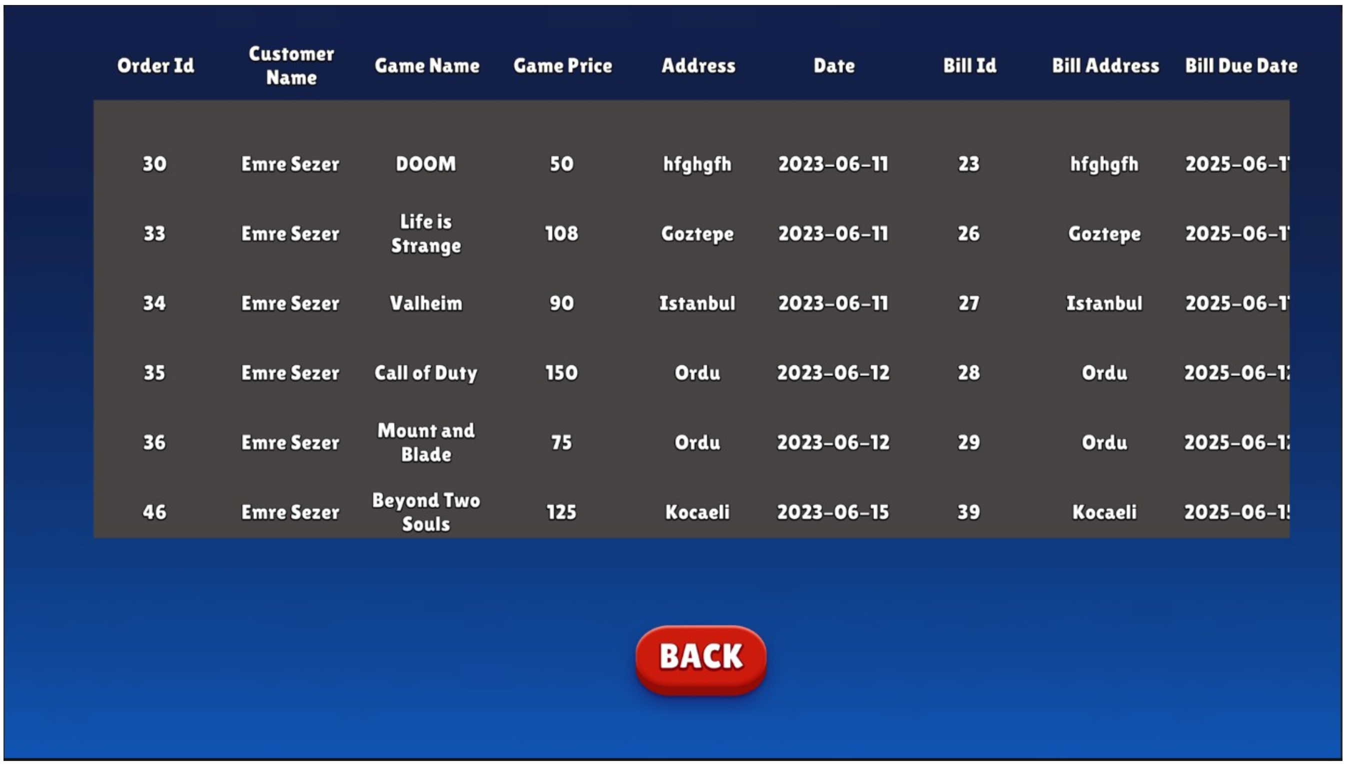
Publisher Update Game Panel:



Error Popup:



Customer View Orders Panel:



Order Id	Customer Name	Game Name	Game Price	Address	Date	Bill Id	Bill Address	Bill Due Date
30	Emre Sezer	DOOM	50	hfghgfh	2023-06-11	23	hfghgfh	2025-06-1
33	Emre Sezer	Life is Strange	108	Goztepe	2023-06-11	26	Goztepe	2025-06-1
34	Emre Sezer	Valheim	90	Istanbul	2023-06-11	27	Istanbul	2025-06-1
35	Emre Sezer	Call of Duty	150	Ordu	2023-06-12	28	Ordu	2025-06-1
36	Emre Sezer	Mount and Blade	75	Ordu	2023-06-12	29	Ordu	2025-06-1
46	Emre Sezer	Beyond Two Souls	125	Kocaeli	2023-06-15	39	Kocaeli	2025-06-1

I developed user interface with Unity Game Engine. Used Php for connecting Unity and MySQL.

Requirements For Testing Project:

I used MySQL and PHP for this project. Php files are stored inside “sqlconnect” folder and sql file is named “1901042640.sql”.

User interface codes are stored inside “Unity Files”. You need to have Unity in order to open this.

You need XAMPP in order to test my project. Open Apache and MySQL servers. Put the “sqlconnect”folder to “xampp/htdocs” on your XAMPP installation folder.

Enter ”http://localhost/phpmyadmin/” at your browser.

Import “1901042640.sql” at the ”http://localhost/phpmyadmin/”.

Run “CSE414-Project.exe” inside the “Executable” folder.