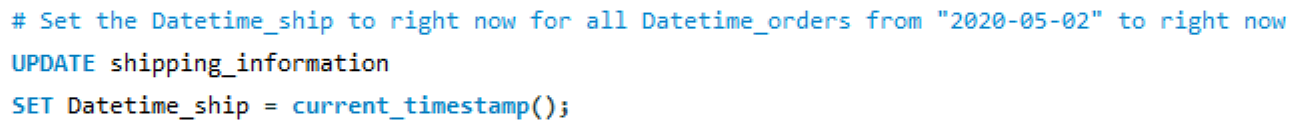
Joseph Edradan
5/3/20
CIS-363-OLH-Enterprise Data Mgmt. w MySQL-CRN 42508

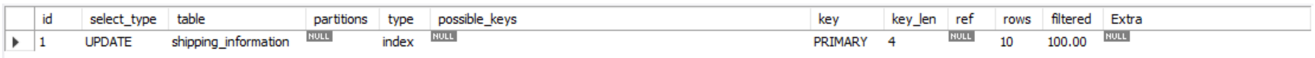
Phase#4 of your Database Application *Database Constraints, Triggers, Views and Stored Procedures*

1. CREATE TABLE statements already made.

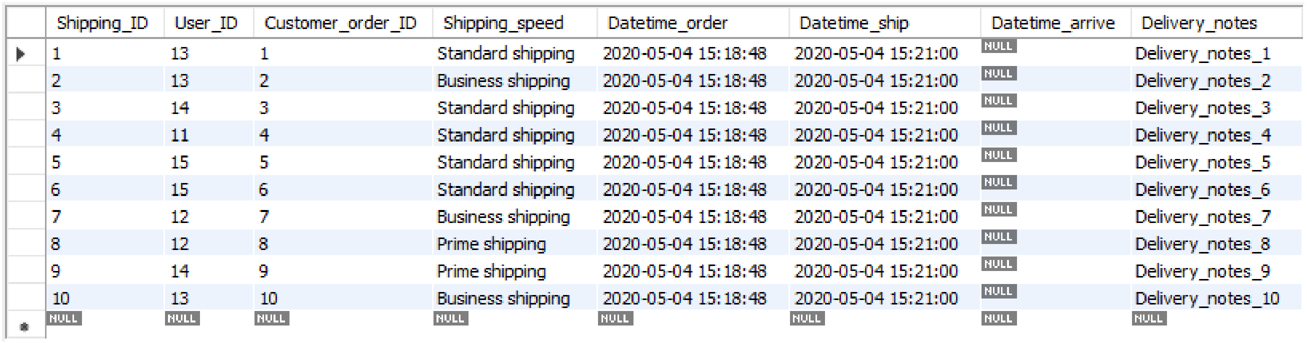
2. Five data modification commands. There isn't a lot of advanced modifications to make here because most rows are unique or that there are many "many to one" relationships to minimize wasted space in the database. Also, there shouldn't be any deletes for transactions because all transactions should be recorded probably for legal and financial reasons.



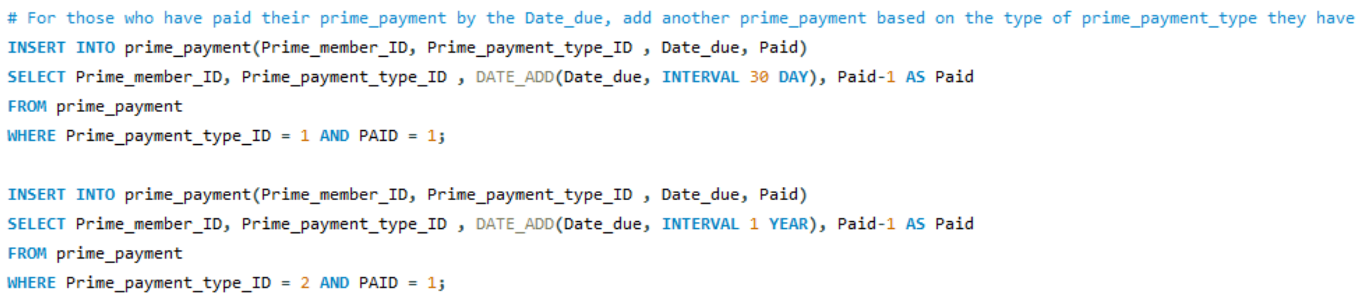
```
# Set the Datetime_ship to right now for all Datetime_orders from "2020-05-02" to right now
UPDATE shipping_information
SET Datetime_ship = current_timestamp();
```



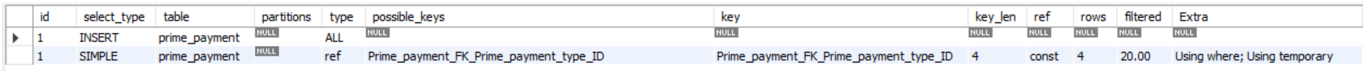
	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	UPDATE	shipping_information	NULL	index	NULL	PRIMARY	4	NULL	10	100.00	NULL



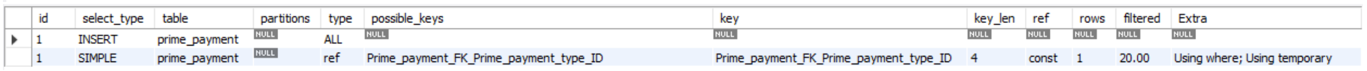
	Shipping_ID	User_ID	Customer_order_ID	Shipping_speed	Datetime_order	Datetime_ship	Datetime_arrive	Delivery_notes
▶	1	13	1	Standard shipping	2020-05-04 15:18:48	2020-05-04 15:21:00	NULL	Delivery_notes_1
	2	13	2	Business shipping	2020-05-04 15:18:48	2020-05-04 15:21:00	NULL	Delivery_notes_2
	3	14	3	Standard shipping	2020-05-04 15:18:48	2020-05-04 15:21:00	NULL	Delivery_notes_3
	4	11	4	Standard shipping	2020-05-04 15:18:48	2020-05-04 15:21:00	NULL	Delivery_notes_4
	5	15	5	Standard shipping	2020-05-04 15:18:48	2020-05-04 15:21:00	NULL	Delivery_notes_5
	6	15	6	Standard shipping	2020-05-04 15:18:48	2020-05-04 15:21:00	NULL	Delivery_notes_6
	7	12	7	Business shipping	2020-05-04 15:18:48	2020-05-04 15:21:00	NULL	Delivery_notes_7
	8	12	8	Prime shipping	2020-05-04 15:18:48	2020-05-04 15:21:00	NULL	Delivery_notes_8
	9	14	9	Prime shipping	2020-05-04 15:18:48	2020-05-04 15:21:00	NULL	Delivery_notes_9
	10	13	10	Business shipping	2020-05-04 15:18:48	2020-05-04 15:21:00	NULL	Delivery_notes_10
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL



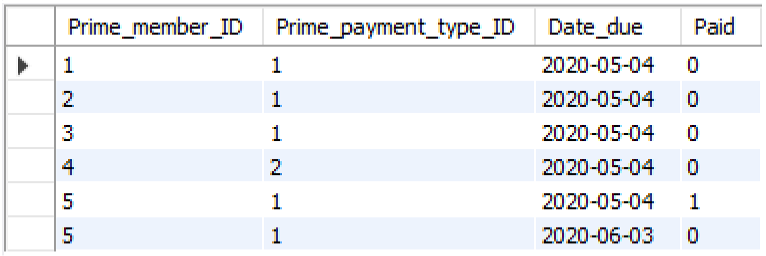
```
# For those who have paid their prime_payment by the Date_due, add another prime_payment based on the type of prime_payment_type they have
INSERT INTO prime_payment(Prime_member_ID, Prime_payment_type_ID , Date_due, Paid)
SELECT Prime_member_ID, Prime_payment_type_ID , DATE_ADD(Date_due, INTERVAL 30 DAY), Paid-1 AS Paid
FROM prime_payment
WHERE Prime_payment_type_ID = 1 AND PAID = 1;

INSERT INTO prime_payment(Prime_member_ID, Prime_payment_type_ID , Date_due, Paid)
SELECT Prime_member_ID, Prime_payment_type_ID , DATE_ADD(Date_due, INTERVAL 1 YEAR), Paid-1 AS Paid
FROM prime_payment
WHERE Prime_payment_type_ID = 2 AND PAID = 1;
```



	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	INSERT	prime_payment	NULL	ALL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
	1	SIMPLE	prime_payment	NULL	ref	Prime_payment_FK_Prime_payment_type_ID	Prime_payment_FK_Prime_payment_type_ID	4	const	4	20.00	Using where; Using temporary

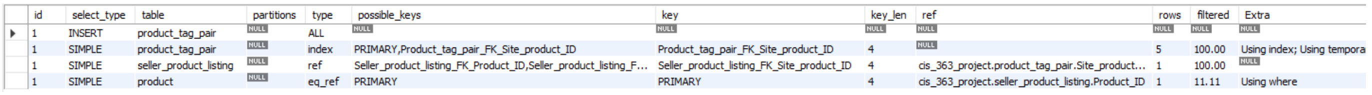


	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	INSERT	prime_payment	NULL	ALL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
	1	SIMPLE	prime_payment	NULL	ref	Prime_payment_FK_Prime_payment_type_ID	Prime_payment_FK_Prime_payment_type_ID	4	const	1	20.00	Using where; Using temporary



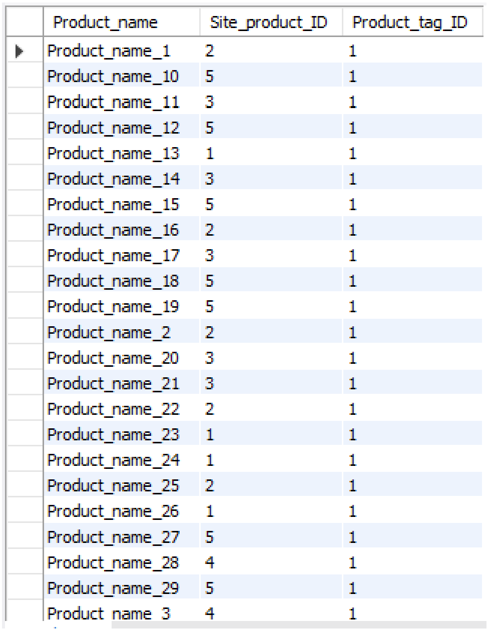
	Prime_member_ID	Prime_payment_type_ID	Date_due	Paid
▶	1	1	2020-05-04	0
	2	1	2020-05-04	0
	3	1	2020-05-04	0
	4	2	2020-05-04	0
	5	1	2020-05-04	1
	5	1	2020-06-03	0

```sql
# All products that have "Product_name" in their name should also have a Product_tage_ID of 1
INSERT INTO product_tag_pair(Site_product_ID, Product_tag_ID)
SELECT temp_table.Site_product_ID as Site_product_ID, 1 as Product_tag_ID FROM (
    SELECT seller_product_listing.Site_product_ID FROM product
    JOIN seller_product_listing ON product.Product_ID = seller_product_listing.Product_ID
    JOIN product_tag_pair ON seller_product_listing.Site_product_ID = product_tag_pair.Site_product_ID
    WHERE Product_name LIKE '%Product_name%'
) as temp_table
ON DUPLICATE KEY UPDATE product_tag_pair.Product_tag_ID = product_tag_pair.Product_tag_ID;
```



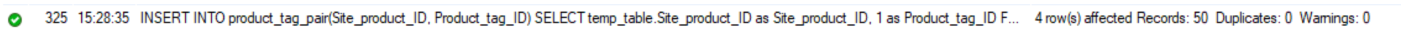
id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	INSERT	product_tag_pair	NULL	ALL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
1	SIMPLE	product_tag_pair	NULL	index	PRIMARY,Product_tag_pair_FK_Site_product_ID	Product_tag_pair_FK_Site_product_ID	4	NULL	5	100.00	Using index; Using tempora
1	SIMPLE	seller_product_listing	NULL	ref	Seller_product_listing_FK_Product_ID,Seller_product_listing_F...	Seller_product_listing_FK_Site_product_ID	4	cis_363_project.product_tag_pair.Site_product...	1	100.00	NULL
1	SIMPLE	product	NULL	eq_ref	PRIMARY	PRIMARY	4	cis_363_project.seller_product_listing.Product_ID	1	11.11	Using where

```sql
SELECT Product_name, seller_product_listing.Site_product_ID, Product_tag_ID FROM product
JOIN seller_product_listing ON product.Product_ID = seller_product_listing.Product_ID
JOIN product_tag_pair ON seller_product_listing.Site_product_ID = product_tag_pair.Site_product_ID
WHERE Product_name LIKE '%Product_name%'
GROUP BY Product_name
ORDER BY Product_name;
```



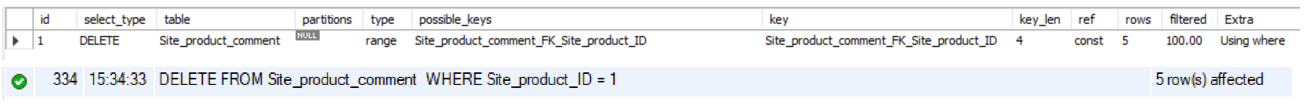
Product_name	Site_product_ID	Product_tag_ID
Product_name_1	2	1
Product_name_10	5	1
Product_name_11	3	1
Product_name_12	5	1
Product_name_13	1	1
Product_name_14	3	1
Product_name_15	5	1
Product_name_16	2	1
Product_name_17	3	1
Product_name_18	5	1
Product_name_19	5	1
Product_name_2	2	1
Product_name_20	3	1
Product_name_21	3	1
Product_name_22	2	1
Product_name_23	1	1
Product_name_24	1	1
Product_name_25	2	1
Product_name_26	1	1
Product_name_27	5	1
Product_name_28	4	1
Product_name_29	5	1
Product_name_3	4	1

(Not all shown)



```
325  15:28:35  INSERT INTO product_tag_pair(Site_product_ID, Product_tag_ID) SELECT temp_table.Site_product_ID as Site_product_ID, 1 as Product_tag_ID F...   4 row(s) affected Records: 50  Duplicates: 0  Warnings: 0
```

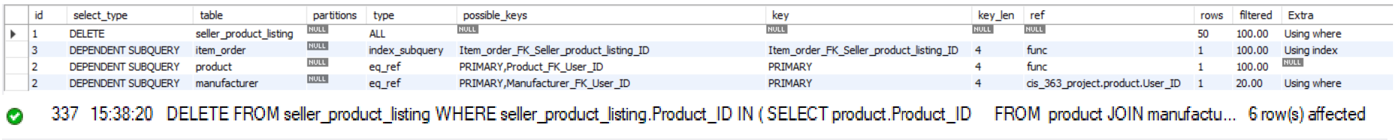
```sql
# Remove comments from Site_product_ID = 1 maybe because of bot comments
DELETE FROM Site_product_comment
WHERE Site_product_ID = 1;
```



id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	DELETE	Site_product_comment	NULL	range	Site_product_comment_FK_Site_product_ID	Site_product_comment_FK_Site_product_ID	4	const	5	100.00	Using where

```
334  15:34:33  DELETE FROM Site_product_comment  WHERE Site_product_ID = 1    5 row(s) affected
```

```sql
# All products made by 'Manufacturer_name_18' regardless of Seller should have their Seller_product_listing removed if they have not been sold already
DELETE FROM seller_product_listing
WHERE seller_product_listing.Product_ID IN (
    SELECT product.Product_ID
    FROM  product
    JOIN manufacturer ON manufacturer.User_ID = product.User_ID
    WHERE manufacturer.Manufacturer_name = "Manufacturer_name_18"
) AND seller_product_listing.Seller_product_listing_ID NOT IN
(
    SELECT Seller_product_listing_ID FROM item_order
);
```

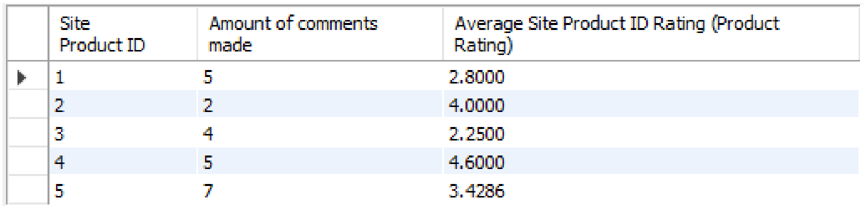


id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	DELETE	seller_product_listing	NULL	ALL	NULL	NULL	NULL	NULL	50	100.00	Using where
3	DEPENDENT SUBQUERY	item_order	NULL	index_subquery	Item_order_FK_Seller_product_listing_ID	Item_order_FK_Seller_product_listing_ID	4	func	1	100.00	Using index
2	DEPENDENT SUBQUERY	product	NULL	eq_ref	PRIMARY,Product_FK_User_ID	PRIMARY	4	func	1	100.00	NULL
2	DEPENDENT SUBQUERY	manufacturer	NULL	eq_ref	PRIMARY,Manufacturer_FK_User_ID	PRIMARY	4	cis_363_project.product.User_ID	1	20.00	Using where

```
337  15:38:20  DELETE FROM seller_product_listing WHERE seller_product_listing.Product_ID IN ( SELECT product.Product_ID   FROM  product JOIN manufactu...  6 row(s) affected
```

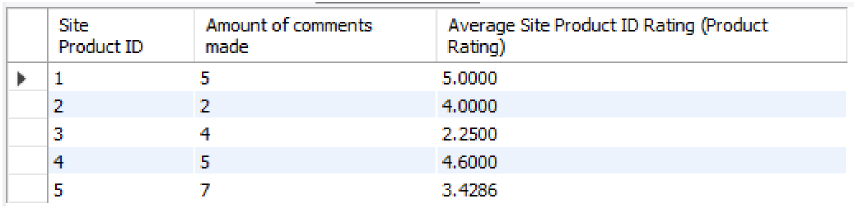
3. Views

```sql
-- Get the average rating for each product on the site as a View
CREATE VIEW All_Site_product_ID_ratings AS
SELECT site_product.Site_product_ID as "Site Product ID", count(Site_product_comment.Site_product_comment_ID) as "Amount of comments made",
AVG(Site_product_comment.product_rating) as "Average Site Product ID Rating (Product Rating)"
FROM site_product
JOIN Site_product_comment ON site_product.site_product_ID = Site_product_comment.site_product_ID
GROUP BY site_product.Site_product_ID;
```



Site Product ID	Amount of comments made	Average Site Product ID Rating (Product Rating)
1	5	2.8000
2	2	4.0000
3	4	2.2500
4	5	4.6000
5	7	3.4286

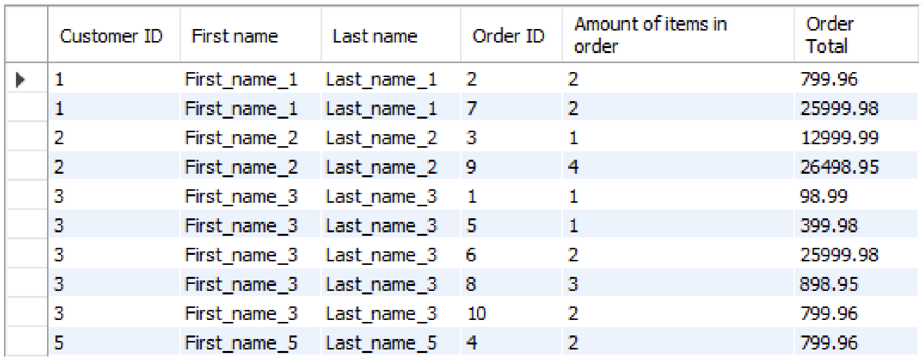
```sql
UPDATE Site_product_comment
SET product_rating = 5
WHERE Site_product_ID = 1;
```



Site Product ID	Amount of comments made	Average Site Product ID Rating (Product Rating)
1	5	5.0000
2	2	4.0000
3	4	2.2500
4	5	4.6000
5	7	3.4286

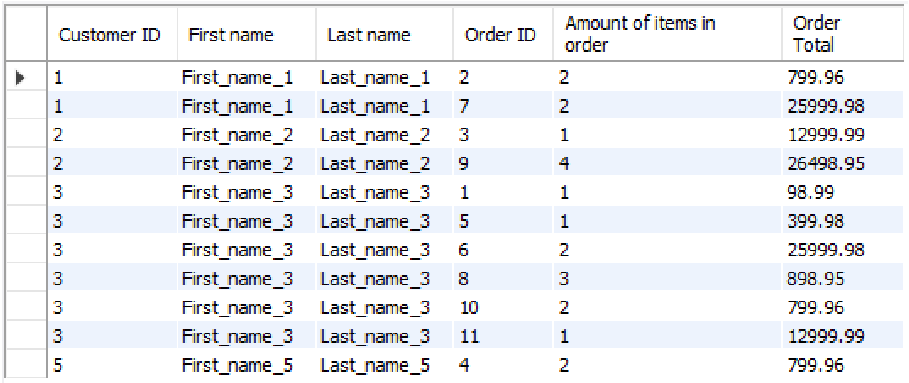
It’s updatable because the SELECT statement is being called on the current instance of the DB.

```
-- Get all Orders made by Customers, the number of items in each order and the total cost of the Order
CREATE VIEW All_Orders_by_Customers AS
SELECT customer.User_ID as "Customer ID" , First_name as "First name", Last_name as "Last name",
customer_order.Customer_order_id as "Order ID",
count(customer_order.Customer_order_id) as "Amount of items in order",
sum(seller_product_listing.product_pricing) as "Order Total"
FROM customer
INNER JOIN customer_order ON customer.User_ID = customer_order.User_ID
INNER JOIN item_order ON customer_order.Customer_order_id = item_order.Customer_order_id
INNER JOIN seller_product_listing on seller_product_listing.seller_product_listing_ID = item_order.seller_product_listing_ID
GROUP BY customer_order.Customer_order_id;
```



Customer ID	First name	Last name	Order ID	Amount of items in order	Order Total
1	First_name_1	Last_name_1	2	2	799.96
1	First_name_1	Last_name_1	7	2	25999.98
2	First_name_2	Last_name_2	3	1	12999.99
2	First_name_2	Last_name_2	9	4	26498.95
3	First_name_3	Last_name_3	1	1	98.99
3	First_name_3	Last_name_3	5	1	399.98
3	First_name_3	Last_name_3	6	2	25999.98
3	First_name_3	Last_name_3	8	3	898.95
3	First_name_3	Last_name_3	10	2	799.96
5	First_name_5	Last_name_5	4	2	799.96

```
INSERT INTO Customer_order (Customer_order_ID, User_ID)
VALUES
('11', '3');

INSERT INTO item_order (Customer_order_ID, Seller_product_listing_ID)
VALUES
('11', '36');
```

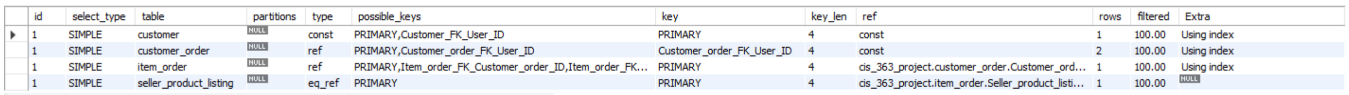


Customer ID	First name	Last name	Order ID	Amount of items in order	Order Total
1	First_name_1	Last_name_1	2	2	799.96
1	First_name_1	Last_name_1	7	2	25999.98
2	First_name_2	Last_name_2	3	1	12999.99
2	First_name_2	Last_name_2	9	4	26498.95
3	First_name_3	Last_name_3	1	1	98.99
3	First_name_3	Last_name_3	5	1	399.98
3	First_name_3	Last_name_3	6	2	25999.98
3	First_name_3	Last_name_3	8	3	898.95
3	First_name_3	Last_name_3	10	2	799.96
3	First_name_3	Last_name_3	11	1	12999.99
5	First_name_5	Last_name_5	4	2	799.96

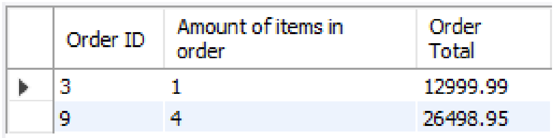
It’s updatable because the SELECT statement is being called on the current instance of the DB.

4. Stored functions / Procedures

```sql
# Get a user's orders (Meant for the user to use)
DROP PROCEDURE IF EXISTS cis_363_project.get_user_all_orders;
DELIMITER //
CREATE PROCEDURE cis_363_project.get_user_all_orders(IN given_user_id INT)
BEGIN
    SELECT
    customer_order.Customer_order_id as "Order ID",
    count(customer_order.Customer_order_id) as "Amount of items in order",
    sum(seller_product_listing.product_pricing) as "Order Total"
    FROM customer
    INNER JOIN customer_order ON customer.User_ID = customer_order.User_ID
    INNER JOIN item_order ON customer_order.Customer_order_id = item_order.Customer_order_id
    INNER JOIN seller_product_listing on seller_product_listing.seller_product_listing_ID = item_order.seller_product_listing_ID
    WHERE customer.User_ID = given_user_id
    GROUP BY customer_order.Customer_order_id;
END //
DELIMITER ;

CALL get_user_all_orders(2);
```

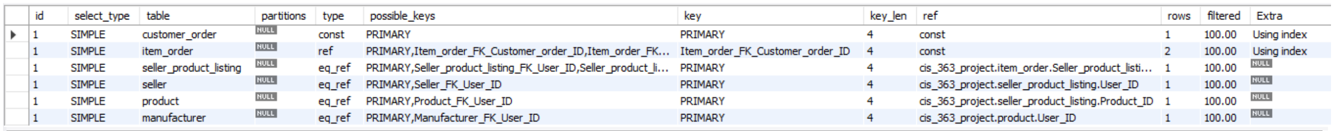


	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	customer	NULL	const	PRIMARY,Customer_FK_User_ID	PRIMARY	4	const	1	100.00	Using index
	1	SIMPLE	customer_order	NULL	ref	PRIMARY,Customer_order_FK_User_ID	Customer_order_FK_User_ID	4	const	2	100.00	Using index
	1	SIMPLE	item_order	NULL	ref	PRIMARY,Item_order_FK_Customer_order_ID,Item_order_FK...	PRIMARY	4	cis_363_project.customer_order.Customer_ord...	1	100.00	Using index
	1	SIMPLE	seller_product_listing	NULL	eq_ref	PRIMARY	PRIMARY	4	cis_363_project.item_order.Seller_product_listi...	1	100.00	NULL

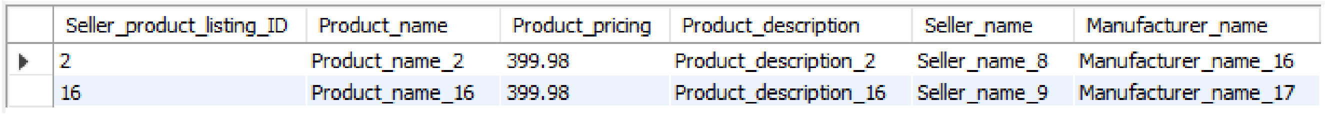


	Order ID	Amount of items in order	Order Total
▶	3	1	12999.99
	9	4	26498.95

```sql
# Get an Order's items
DROP PROCEDURE IF EXISTS cis_363_project.get_items_in_custoemr_order_id;
DELIMITER //
CREATE PROCEDURE cis_363_project.get_items_in_custoemr_order_id(IN given_customer_order_id INT)
BEGIN
    SELECT seller_product_listing.Seller_product_listing_ID, product.Product_name, seller_product_listing.Product_pricing, seller_product_listing.Product_description,
    seller.Seller_name, manufacturer.Manufacturer_name
    FROM customer_order
    JOIN item_order ON item_order.Customer_order_ID = customer_order.Customer_order_ID
    JOIN seller_product_listing ON seller_product_listing.Seller_product_listing_ID = item_order.Seller_product_listing_ID
    JOIN seller ON seller.User_ID = seller_product_listing.User_ID
    JOIN product ON product.product_ID = seller_product_listing.product_ID
    JOIN manufacturer ON manufacturer.User_ID = product.User_ID
    WHERE customer_order.Customer_order_ID = given_customer_order_id;
END //
DELIMITER ;

CALL get_items_in_custoemr_order_id(2);
```



	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	customer_order	NULL	const	PRIMARY	PRIMARY	4	const	1	100.00	Using index
	1	SIMPLE	item_order	NULL	ref	PRIMARY,Item_order_FK_Customer_order_ID,Item_order_FK...	Item_order_FK_Customer_order_ID	4	const	2	100.00	Using index
	1	SIMPLE	seller_product_listing	NULL	eq_ref	PRIMARY,Seller_product_listing_FK_User_ID,Seller_product_li...	PRIMARY	4	cis_363_project.item_order.Seller_product_listi...	1	100.00	NULL
	1	SIMPLE	seller	NULL	eq_ref	PRIMARY,Seller_FK_User_ID	PRIMARY	4	cis_363_project.seller_product_listing.User_ID	1	100.00	NULL
	1	SIMPLE	product	NULL	eq_ref	PRIMARY,Product_FK_User_ID	PRIMARY	4	cis_363_project.seller_product_listing.Product_ID	1	100.00	NULL
	1	SIMPLE	manufacturer	NULL	eq_ref	PRIMARY,Manufacturer_FK_User_ID	PRIMARY	4	cis_363_project.product.User_ID	1	100.00	NULL

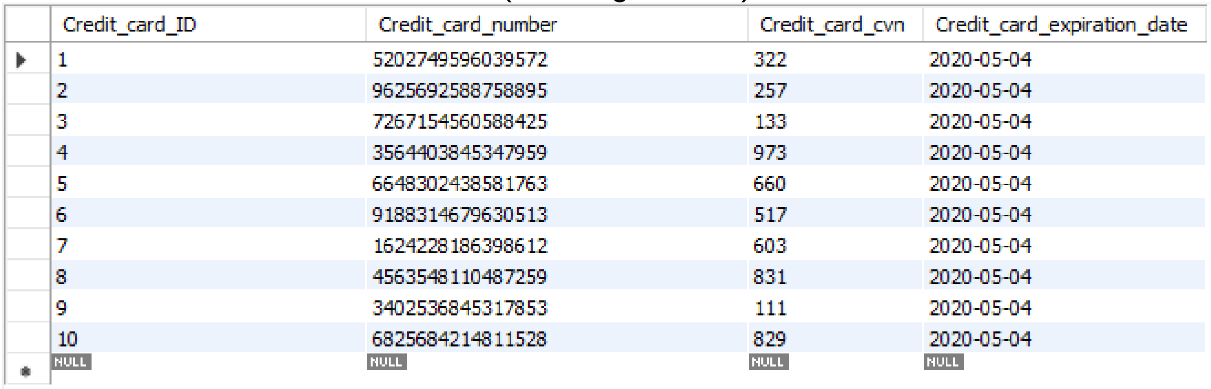


	Seller_product_listing_ID	Product_name	Product_pricing	Product_description	Seller_name	Manufacturer_name
▶	2	Product_name_2	399.98	Product_description_2	Seller_name_8	Manufacturer_name_16
	16	Product_name_16	399.98	Product_description_16	Seller_name_9	Manufacturer_name_17

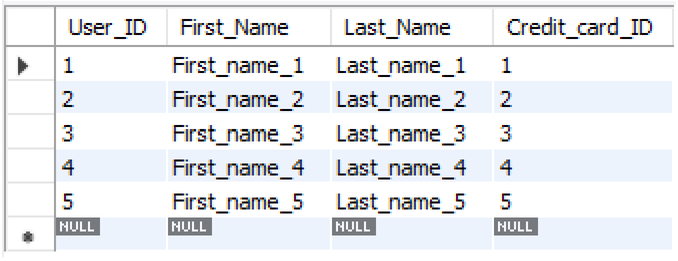
5. Triggers are VERY limited and *MySql* does not support INSTEAD OF

```
-- DROP PROCEDURE IF EXISTS cis_363_project.delete_credit_card;
# Delete routine for trigger_customer_update_credit_card and trigger_seller_update_credit_card
DELIMITER //
CREATE PROCEDURE cis_363_project.delete_credit_card(IN Credit_card_ID_OLD INT, IN Credit_card_ID_NEW INT)
BEGIN
    IF (Credit_card_ID_NEW != Credit_card_ID_OLD) THEN
        DELETE FROM credit_card
        WHERE credit_card.Credit_card_ID = Credit_card_ID_OLD;
    END IF;
END //
DELIMITER ;

-- DROP TRIGGER IF EXISTS cis_363_project.trigger_customer_update_credit_card;
# Delete the credit_card row when a Customer deletes their Credit_card_ID
DELIMITER //
CREATE TRIGGER cis_363_project.trigger_customer_update_credit_card
AFTER UPDATE ON cis_363_project.customer
FOR EACH ROW
BEGIN
--  IF (NEW.Credit_card_ID != OLD.Credit_card_ID) THEN
--      DELETE FROM credit_card
--          WHERE credit_card.Credit_card_ID = OLD.Credit_card_ID;
--  END IF;
    CALL delete_credit_card(OLD.Credit_card_ID, NEW.Credit_card_ID);

END //
DELIMITER ;

-- DROP TRIGGER IF EXISTS cis_363_project.trigger_seller_update_credit_card;
# Delete the credit_card row when a Seller deletes their Credit_card_ID
DELIMITER //
CREATE TRIGGER cis_363_project.trigger_seller_update_credit_card
AFTER UPDATE ON cis_363_project.seller
FOR EACH ROW
BEGIN
    CALL delete_credit_card(OLD.Credit_card_ID, NEW.Credit_card_ID);
END //
DELIMITER ;
# TESTING ZONE Working version
INSERT INTO credit_card VALUES ('11', '5500282033727105', '474', '2020-05-04');
INSERT INTO credit_card VALUES ('12', '5500282033727105', '474', '2020-05-04');

UPDATE customer
SET customer.Credit_card_ID = 11
WHERE customer.User_ID = 1;

# TESTING ZONE Error version
UPDATE customer
SET customer.Credit_card_ID = 4
WHERE customer.User_ID = 1;

UPDATE customer
SET customer.Credit_card_ID = 2
WHERE customer.User_ID = 1;
```

TESTING ZONE Before Execution (Working version)

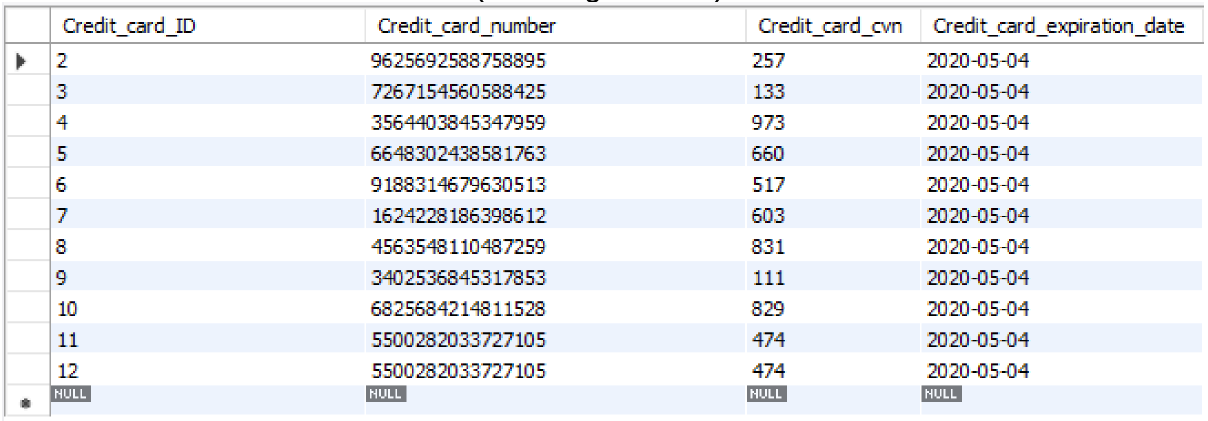


Credit_card_ID	Credit_card_number	Credit_card_cvn	Credit_card_expiration_date
1	5202749596039572	322	2020-05-04
2	9625692588758895	257	2020-05-04
3	7267154560588425	133	2020-05-04
4	3564403845347959	973	2020-05-04
5	6648302438581763	660	2020-05-04
6	9188314679630513	517	2020-05-04
7	1624228186398612	603	2020-05-04
8	4563548110487259	831	2020-05-04
9	3402536845317853	111	2020-05-04
10	6825684214811528	829	2020-05-04
NULL	NULL	NULL	NULL

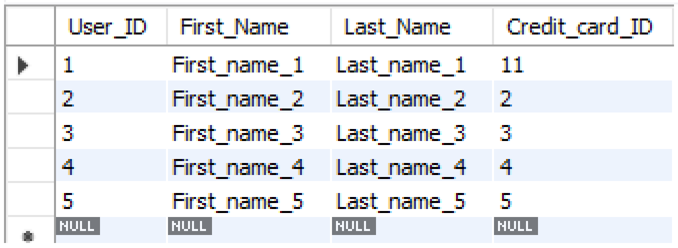


User_ID	First_Name	Last_Name	Credit_card_ID
1	First_name_1	Last_name_1	1
2	First_name_2	Last_name_2	2
3	First_name_3	Last_name_3	3
4	First_name_4	Last_name_4	4
5	First_name_5	Last_name_5	5
NULL	NULL	NULL	NULL

TESTING ZONE After Execution (Working version)

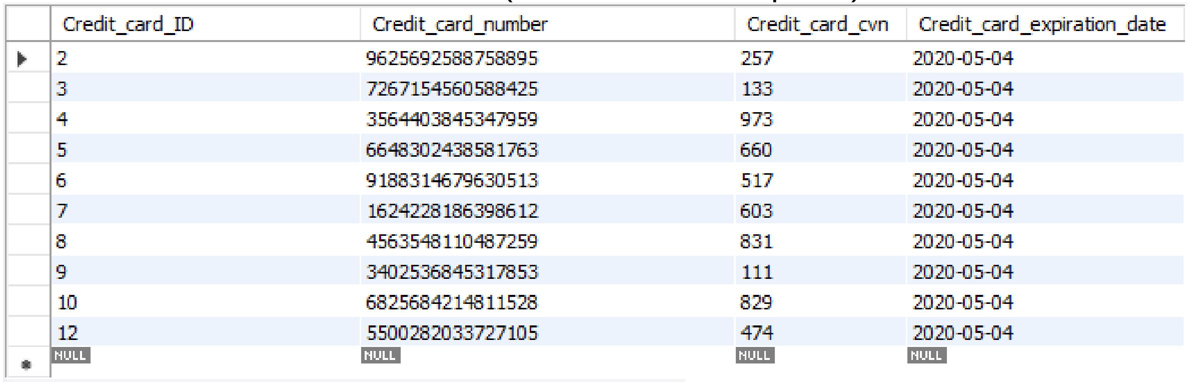


Credit_card_ID	Credit_card_number	Credit_card_cvn	Credit_card_expiration_date
2	9625692588758895	257	2020-05-04
3	7267154560588425	133	2020-05-04
4	3564403845347959	973	2020-05-04
5	6648302438581763	660	2020-05-04
6	9188314679630513	517	2020-05-04
7	1624228186398612	603	2020-05-04
8	4563548110487259	831	2020-05-04
9	3402536845317853	111	2020-05-04
10	6825684214811528	829	2020-05-04
11	5500282033727105	474	2020-05-04
12	5500282033727105	474	2020-05-04
NULL	NULL	NULL	NULL

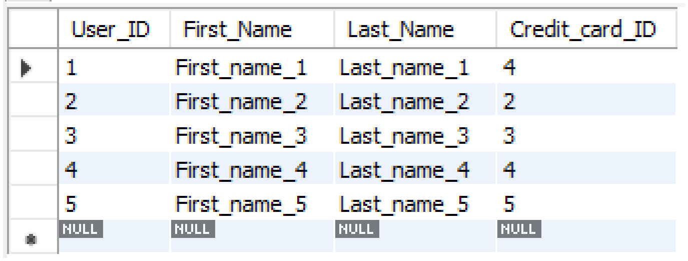


User_ID	First_Name	Last_Name	Credit_card_ID
1	First_name_1	Last_name_1	11
2	First_name_2	Last_name_2	2
3	First_name_3	Last_name_3	3
4	First_name_4	Last_name_4	4
5	First_name_5	Last_name_5	5
NULL	NULL	NULL	NULL

TESTING ZONE Before Execution (Error version 1st Update)

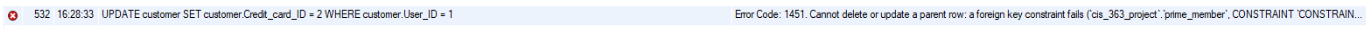


Credit_card_ID	Credit_card_number	Credit_card_cvn	Credit_card_expiration_date
2	9625692588758895	257	2020-05-04
3	7267154560588425	133	2020-05-04
4	3564403845347959	973	2020-05-04
5	6648302438581763	660	2020-05-04
6	9188314679630513	517	2020-05-04
7	1624228186398612	603	2020-05-04
8	4563548110487259	831	2020-05-04
9	3402536845317853	111	2020-05-04
10	6825684214811528	829	2020-05-04
12	5500282033727105	474	2020-05-04
NULL	NULL	NULL	NULL



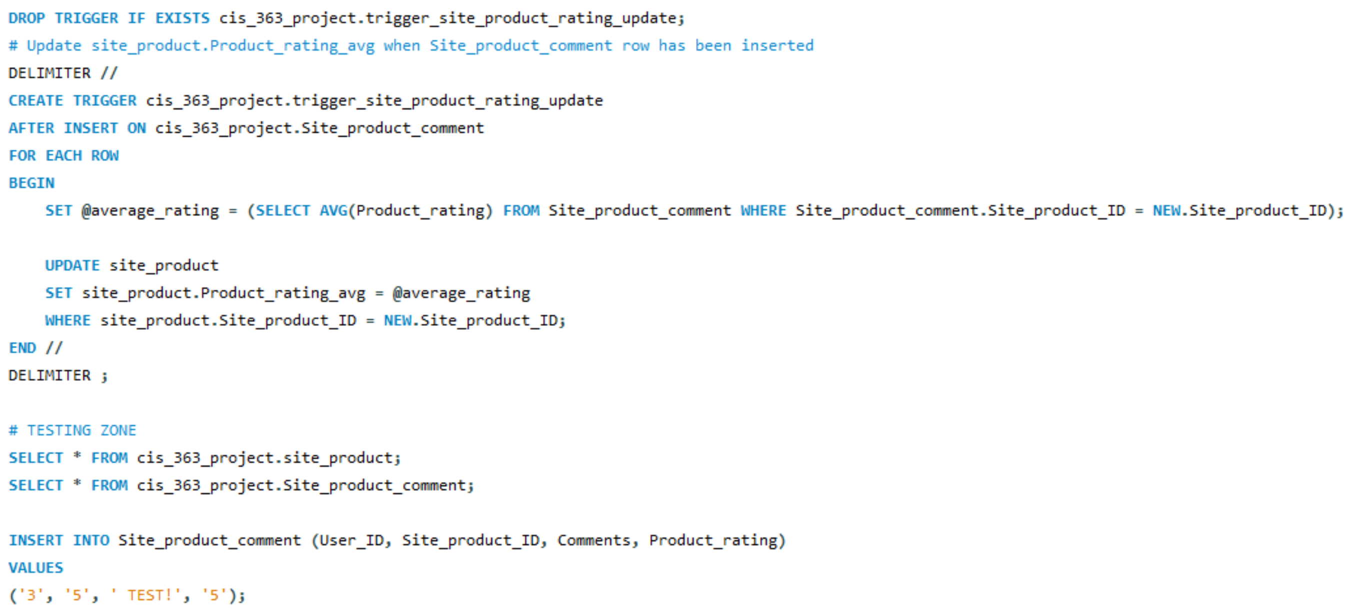
User_ID	First_Name	Last_Name	Credit_card_ID
1	First_name_1	Last_name_1	4
2	First_name_2	Last_name_2	2
3	First_name_3	Last_name_3	3
4	First_name_4	Last_name_4	4
5	First_name_5	Last_name_5	5
NULL	NULL	NULL	NULL

TESTING ZONE After Execution (Error version 2nd Update)



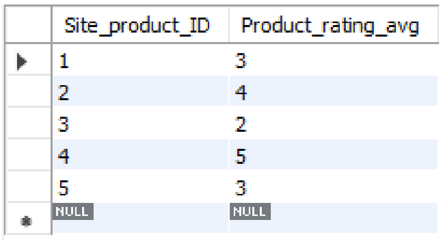
```
532  16:28:33  UPDATE customer SET customer.Credit_card_ID = 2 WHERE customer.User_ID = 1  Error Code: 1451. Cannot delete or update a parent row: a foreign key constraint fails (`cis_363_project`.`prime_member`, CONSTRAINT `CONSTRAIN...
```

This should be an Error because the Credit_card_ID 4 is already tied to User_ID 4 meaning that you can’t delete a credit_card row that has it’s foreign key existing somewhere else. You should not loop in the Trigger to see if the key Credit_card_ID has it’s foreign key somewhere else because that is a security issue and will be costly on the DB if the DB is big. To fix this, Update again setting Credit_card_ID = null then do the correct update. However, this issue should have never happened in the first place because Credit_card_ID should not be set up manually and instead is automatically incremented.



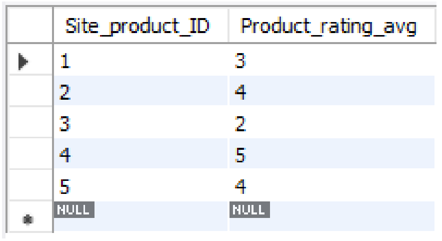
```
DROP TRIGGER IF EXISTS cis_363_project.trigger_site_product_rating_update;
# Update site_product.Product_rating_avg when Site_product_comment row has been inserted
DELIMITER //
CREATE TRIGGER cis_363_project.trigger_site_product_rating_update
AFTER INSERT ON cis_363_project.Site_product_comment
FOR EACH ROW
BEGIN
    SET @average_rating = (SELECT AVG(Product_rating) FROM Site_product_comment WHERE Site_product_comment.Site_product_ID = NEW.Site_product_ID);

    UPDATE site_product
    SET site_product.Product_rating_avg = @average_rating
    WHERE site_product.Site_product_ID = NEW.Site_product_ID;
END //
DELIMITER ;

# TESTING ZONE
SELECT * FROM cis_363_project.site_product;
SELECT * FROM cis_363_project.Site_product_comment;

INSERT INTO Site_product_comment (User_ID, Site_product_ID, Comments, Product_rating)
VALUES
('3', '5', ' TEST!', '5');
```

TESTING ZONE Before Execution



Site_product_ID	Product_rating_avg
1	3
2	4
3	2
4	5
5	3
NULL	NULL

TESTING ZONE After Execution



Site_product_ID	Product_rating_avg
1	3
2	4
3	2
4	5
5	4
NULL	NULL

Site_product has Product_rating_avg and is only updated when a Site_product_comment is inserted to prevent the DB from excessively calculating a product’s avg rating such as in the VIEW “All_Site_product_ID_ratings”.