Assignment 4: Normalization

Submission by: Abhishek Nair (NUID: 002724762) and Hrishikesh Sanjay Pawar (NUID: 002707307)

Topic: University Marketplace Database

Topic description:

When Abhishek and I arrived in Boston for our Master's program, it was an arduous task among 500+ students to find used essential items like study tables, chairs, bed frames, study lamps, etc. University Marketplace Database project is a database of used or pre-owned essential items for peer to peer buying/selling created for university students. The project addresses two issues:

1. Mostly buying/selling of used items is done on instant messengers like Whatsapp via multiple groups. This creates a decentralized environment where it is difficult to keep track of all the buying/selling opportunities.
2. On general platforms like Facebook Marketplace or Ebay, the buyers/sellers might not be trustworthy and/or may be located at an inconvenient distance from the university.

We are trying to resolve these issues by creating a centralized database of pre-owned essential items that can be accessed at a single place. The marketplace will be segregated based on universities to make sure that buyers and sellers are located at a reasonable distance. We will be using users' university email address for the verification purpose. Along with the general details of the product, distance of each product from the buyers will be computed based on the lag-long coordinates. This will help buyers decide which seller to buy from. We are also planning to incorporate a premium subscription feature into the project for sellers. This subscription will enable the sellers to list their products at the top of the search results.

EVALUATING TABLES FOR NORMALIZATION FORMS

University table :

First NF -

Consists of primary key: university_id.

Values in each column are atomic.

There are no two columns that store similar information.

This table meets all the requirements for the first normal form.

Second NF -

All the requirements of 1st Normal form are met.

All the columns have full functional dependency on the primary key university_id, so there are no partial dependencies.

There are no columns having calculated data.

Thus, the table meets the requirements for second normal form.

Third NF –

All the requirements for 2nd Normal form are met.

There are no transitive dependencies present in the table.

Thus, this table is established to be in the third normal form.

Student table :

First NF -

Consists of primary key: student_id.

Values in each column are atomic.

There are no two columns that store similar information.

This table meets all the requirements for the first normal form.

Second NF -

All the requirements of 1st Normal form are met.

All the columns have full functional dependency on the primary key student_id, so there are no partial dependencies.

There are no columns having calculated data.

Thus, the table meets the requirements for second normal form.

Third NF –

All the requirements for 2nd Normal form are met.

There are no transitive dependencies present in the table.

Thus, this table is established to be in the third normal form.

Buyer table :

First NF -

Consists of primary key: buyer_id.

Values in each column are atomic.

There are no two columns that store similar information.

This table meets all the requirements for the first normal form.

Second NF -

All the requirements of 1st Normal form are met.

All the columns have full functional dependency on the primary key buyer_id, so there are no partial dependencies.

There are no columns having calculated data.

Thus, the table meets the requirements for second normal form.

Third NF –

All the requirements for 2nd Normal form are met.

There are no transitive dependencies present in the table.

Thus, this table is established to be in the third normal form.

Seller table :

First NF -

Consists of primary key: seller_id.

Values in each column are atomic.

There are no two columns that store similar information.

This table meets all the requirements for the first normal form.

Second NF -

All the requirements of 1st Normal form are met.

All the columns have full functional dependency on the primary key seller_id, so there are no partial dependencies.

There are no columns having calculated data.

Thus, the table meets the requirements for second normal form.

Third NF –

All the requirements for 2nd Normal form are met.

There are no transitive dependencies present in the table.

Thus, this table is established to be in the third normal form.

Twitter_Order table :

First NF -

The primary key for this table t_order_id may not have unique values hence failing to be considered a primary key. This is because there might be orders having multiple products in which case there would be more than one record for each order leading to duplication of the field t_order_id. Thus, the first normal form is violated.

To tackle this problem, we have divided the table into two separate tables: Twitter_Order_Details table and Twitter_Order_Header table. These tables have been created in such a way that the order line records are stored in the Order_Details table, so this table may have multiple records for a single order depending on the number of unique products purchased in this order. However, the Order_Header table will have a single record for each of the order representing the details for each of the order.

Tweet table:

First NF -

Consists of primary key: tweet_id.

Values in each column are atomic.

There are no two columns that store similar information.

This table meets all the requirements for the first normal form.

Second NF -

All the requirements of 1st Normal form are met.

All the columns have full functional dependency on the primary key tweet_id, so there are no partial dependencies.

There are no columns having calculated data.

Thus, the table meets the requirements for second normal form.

Third NF –

All the requirements for 2nd Normal form are met.

There are no transitive dependencies present in the table.

Thus, this table is established to be in the third normal form.

Tweet_Tags table:

First NF -

Tweet_Tags table does not consist of a primary key. It only has a foreign key. Thus, it currently violates the first normal form. A primary key field tweet_tag_id has been added to this table.

Values in each column are atomic.

There are no two columns that store similar information.

This table now meets all the requirements for the first normal form.

Second NF -

All the requirements of 1st Normal form are met.

All the columns have full functional dependency on the primary key tweet_id, so there are no partial dependencies.

There are no columns having calculated data.

Thus, the table meets the requirements for second normal form.

Third NF –

All the requirements for 2nd Normal form are met.

There are no transitive dependencies present in the table.

Thus, this table is established to be in the third normal form.

Tweet_Mentions table:

First NF -

Tweet_Mentions table does not consist of a primary key. It only has a foreign key. Thus, it currently violates the first normal form. A primary key field tweet_mentions_id has been added to this table.

Values in each column are atomic.

There are no two columns that store similar information.

This table now meets all the requirements for the first normal form.

Second NF -

All the requirements of 1st Normal form are met.

All the columns have full functional dependency on the primary key tweet_mentions_id, so there are no partial dependencies.

There are no columns having calculated data.

Thus, the table meets the requirements for second normal form.

Third NF –

All the requirements for 2nd Normal form are met.

There are no transitive dependencies present in the table.

Thus, this table is established to be in the third normal form.

Product table:

First NF -

Consists of primary key: product_id.

Values in each column are atomic.

There are no two columns that store similar information.

This table meets all the requirements for the first normal form.

Second NF -

All the requirements of 1st Normal form are met.

All the columns have full functional dependency on the primary key product_id, so there are no partial dependencies.

There are no columns having calculated data.

Thus, the table meets the requirements for second normal form.

Third NF –

All the requirements for 2nd Normal form are met.

There are no transitive dependencies present in the table.

Thus, this table is established to be in the third normal form.

Feedback table:

First NF -

Consists of primary key: feedback_id.

Values in each column are atomic.

There are no two columns that store similar information.

This table meets all the requirements for the first normal form.

Second NF -

All the requirements of 1st Normal form are met.

All the columns have full functional dependency on the primary key feedback_id, so there are no partial dependencies.

There are no columns having calculated data.

Thus, the table meets the requirements for second normal form.

Third NF –

All the requirements for 2nd Normal form are met.

There are no transitive dependencies present in the table.

Thus, this table is established to be in the third normal form.

Category table:

First NF -

Consists of primary key: category_id.

Values in each column are atomic.

There are no two columns that store similar information.

This table meets all the requirements for the first normal form.

Second NF -

All the requirements of 1st Normal form are met.

All the columns have full functional dependency on the primary key category_id, so there are no partial dependencies.

There are no columns having calculated data.

Thus, the table meets the requirements for second normal form.

Third NF –

All the requirements for 2nd Normal form are met.

There are no transitive dependencies present in the table.

Thus, this table is established to be in the third normal form.

RESTRUCTURING THE DATABASE FOR NORMALIZATION TO 3rd NF

1. **SQL creating a new table for twitter_order_header:**

```
CREATE TABLE `twitter_schema`.`twitter_order_header` (
  `t_order_id` INT NOT NULL,
  `buyer_id` INT NOT NULL,
  `seller_id` INT NOT NULL,
  `tweet_id` INT NOT NULL,
  `feedback_id` INT NOT NULL,
  `order_date` DATE NOT NULL,
  PRIMARY KEY (`t_order_id`));
```

2. **SQL creating a new table for twitter_order_details:**

```
CREATE TABLE `twitter_schema`.`twitter_order_details` (
  `t_order_details_id` INT NOT NULL,
  `t_order_id` INT NOT NULL,
  `product_id` INT NOT NULL,
  `price` FLOAT NOT NULL,
  PRIMARY KEY (`t_order_details_id`),
  INDEX `t_order_id_fk_idx` (`t_order_id` ASC) VISIBLE,
  CONSTRAINT `t_order_id_fk`
    FOREIGN KEY (`t_order_id`)
    REFERENCES `twitter_schema`.`twitter_order_header` (`t_order_id`));
```

3. **SQL altering the table tweet_mentions to add a primary key:**

```
ALTER TABLE `twitter_schema`.`tweet_mentions`
ADD COLUMN `tweet_mentions_id` INT NOT NULL,
ADD PRIMARY KEY (`tweet_mentions_id`);
```

4. **SQL altering the table tweet_tags to add a primary key:**

```
ALTER TABLE `twitter_schema`.`tweet_tags`
ADD COLUMN `tweet_tag_id` INT NOT NULL ,
ADD PRIMARY KEY (`tweet_tag_id`);
```

SQL creating rest of the normalized database:

```
CREATE TABLE `buyer` (
  `buyer_id` int NOT NULL,
  `student_id` int NOT NULL,
  PRIMARY KEY (`buyer_id`),
  KEY `student_id_fk_idx` (`student_id`),
  CONSTRAINT `student_id_fk` FOREIGN KEY (`student_id`) REFERENCES `student` (`student_id`));
```

```
CREATE TABLE `category` (
  `category_id` int NOT NULL,
  `category_name` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`category_id`));
```

```
CREATE TABLE `feedback` (
  `feedback_id` int NOT NULL,
  `feedback_score` int NOT NULL,
  `comments` varchar(200) DEFAULT NULL,
  PRIMARY KEY (`feedback_id`));


CREATE TABLE `product` (
  `product_id` int NOT NULL,
  `product_name` varchar(100) NOT NULL,
  `category_id` int DEFAULT NULL,
  PRIMARY KEY (`product_id`),
  KEY `category_fk_idx` (`category_id`),
  CONSTRAINT `category_id_fk` FOREIGN KEY (`category_id`) REFERENCES `category`
(`category_id`));


CREATE TABLE `seller` (
  `seller_id` int NOT NULL,
  `student_id` int NOT NULL,
  `premium_flag` int NOT NULL,
  PRIMARY KEY (`seller_id`),
  KEY `student_id_fk_idx` (`student_id`),
  CONSTRAINT `student_id_fk_seller` FOREIGN KEY (`student_id`) REFERENCES `student`
(`student_id`));
```

```
CREATE TABLE `student` (
  `student_id` int NOT NULL,
  `university_id` int NOT NULL,
  `first_name` varchar(100) NOT NULL,
  `last_name` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`student_id`),
  KEY `university_id_fk_idx` (`university_id`),
  CONSTRAINT `university_id_fk` FOREIGN KEY (`university_id`) REFERENCES `university` (`university_id`));
```

```
CREATE TABLE `tweet` (
  `tweet_id` int NOT NULL AUTO_INCREMENT,
  `twitter_handle` varchar(50) NOT NULL,
  `tweet_text` varchar(250) NOT NULL,
  `tweet_date` date NOT NULL,
  `profile_image_url` varchar(300) DEFAULT NULL,
  `user_created_at` date NOT NULL,
  `retweets` int NOT NULL,
  PRIMARY KEY (`tweet_id`));
```

```
CREATE TABLE `tweet_mentions` (
  `tweet_id` int NOT NULL AUTO_INCREMENT,
  `source_user` varchar(100) NOT NULL,
  `target_user` varchar(100) NOT NULL,
  `tweet_mentions_id` int NOT NULL,
  PRIMARY KEY (`tweet_mentions_id`),
  KEY `tweet_id_fk_mentions_idx` (`tweet_id`),
  CONSTRAINT `tweet_mentions_fk` FOREIGN KEY (`tweet_id`) REFERENCES `tweet` (`tweet_id`));


CREATE TABLE `tweet_tags` (
  `tweet_id` int NOT NULL AUTO_INCREMENT,
  `tag` varchar(20) NOT NULL,
  `target_user` varchar(100) NOT NULL,
  `tweet_tag_id` int NOT NULL,
  PRIMARY KEY (`tweet_tag_id`),
  KEY `twitter_id_fk_tags_idx` (`tweet_id`),
  CONSTRAINT `tweet_id_fk_tags` FOREIGN KEY (`tweet_id`) REFERENCES `tweet` (`tweet_id`));


CREATE TABLE `university` (
  `university_id` int NOT NULL,
  `university_name` varchar(100) NOT NULL,
  `state` varchar(100) NOT NULL,
  `city` varchar(100) NOT NULL,
  PRIMARY KEY (`university_id`));
```

VIEWS CREATED FOR ALL THE USE CASE QUERIES

1) Use Case 1: Top 3 orders which have received the negative feedback

```
CREATE VIEW  twitter_schema.TOP_3_Negative_feedback_orders AS
SELECT t.t_order_id, f.feedback_score
FROM twitter_schema.twitter_order_header t
INNER JOIN twitter_schema.Feedback f
ON f.feedback_id = t.feedback_id
ORDER BY feedback_score ASC
LIMIT 3;
```

2) Use Case 2: View Category Name and Product Name for a specific product ID

```
CREATE VIEW  twitter_schema.View_Category_Name_and_Product_name_for_product_id
AS
SELECT p.product_name, c.category_name
FROM twitter_schema.Product p
INNER JOIN twitter_schema.Category c
ON p.category_id = c.category_id;
```

3) Use Case 3: View tags mentioned by the particular twitter_user in a tweet_text

```
CREATE VIEW  twitter_schema.View_tags_mentioned_by_the_particular_twitter_user AS
SELECT t.twitter_handle, t.tweet_text, t.tweet_date, tt.tag
FROM twitter_schema.Tweet t
INNER JOIN twitter_schema.Tweet_tags tt
ON t.tweet_id = tt.tweet_id;
```

4) Use Case 4: Who are the sellers that are students and have they enrolled for a premium option?

```
CREATE VIEW
twitter_schema.sellers_that_are_students_who_have_enrolled_for_premium_option AS
SELECT student.first_name, student.last_name, seller.premium_flag
FROM twitter_schema.Student student
INNER JOIN twitter_schema.Seller seller
ON student.student_id = seller.student_id
where seller.premium_flag = 1;
```

5) Use Case 5: Who is the source and the target user mentioned by the particular twitter user in a tweet

```
CREATE VIEW
twitter_schema.source_and_target_user_mentioned_by_the_particular_twitter_user AS
SELECT t.twitter_handle, t.tweet_text, t.tweet_date, tm.source_user, tm.target_user
FROM twitter_schema.Tweet t
INNER JOIN twitter_schema.Tweet_Mentions tm
ON t.tweet_id = tm.tweet_id;
```

6) Use Case 6: View the items that can be purchased by the buyer

```
CREATE VIEW  twitter_schema.view_the_items_that_can_be_purchased_by_the_buyer AS
SELECT p.product_name, c.category_name
FROM twitter_schema.Product p
INNER JOIN twitter_schema.Category c
ON c.category_id = p.category_id;
```

7) Use Case 7: Which seller from the University sold the most items

```
CREATE VIEW  twitter_schema.which_seller_from_the_university_sold_most_items AS
SELECT  a.first_name, a.last_name, t.Number_of_items_sold
FROM
(
        SELECT seller_id, count(product_id) AS `Number_of_items_sold`
        FROM twitter_schema.twitter_order_header oh
        INNER JOIN twitter_schema.twitter_order_details od
        ON oh.t_order_id = od.t_order_id
        GROUP BY seller_id
        ORDER BY 2 desc
        LIMIT 1
) AS t INNER JOIN twitter_schema.Seller b
 ON t.seller_id = b.seller_id
 INNER JOIN twitter_schema.Student a
 ON a.student_id = b.student_id;
```

8) Use Case 8: View a product below a particular price

```
CREATE VIEW  twitter_schema.view_product_below_particular_price AS
SELECT p.product_name
FROM twitter_schema.Product p
INNER JOIN twitter_schema.twitter_order_details t
ON p.product_id = t.product_id
WHERE t.price < 5000;
```

9) Use Case 9: Top 3 orders which received the worst feedback

```
CREATE VIEW  twitter_schema.top_3_orders_received_worst_feedback AS
SELECT t.t_order_id
FROM twitter_schema.twitter_order_header t
INNER JOIN twitter_schema.Feedback f
ON f.feedback_id = t.feedback_id
ORDER BY feedback_score ASC
LIMIT 3;
```

10) Use Case 10: Top 3 orders which have received the positive feedback

```
CREATE VIEW  twitter_schema.top_3_orders_received_positive_feedback AS
SELECT t.t_order_id
FROM twitter_schema.twitter_order_header t
INNER JOIN twitter_schema.Feedback f
ON f.feedback_id = t.feedback_id
ORDER BY feedback_score DESC
LIMIT 3;
```

11) Use Case 11: View the product with highest price.

```
CREATE VIEW twitter_schema.product_with_the_highest_price AS
SELECT product_name, Price
FROM
(SELECT product_id, MAX(price) Price
FROM twitter_schema.twitter_order_header oh
INNER JOIN twitter_schema.twitter_order_details od
ON oh.t_order_id = od.t_order_id
GROUP BY 1
ORDER BY Price Desc
LIMIT 1) O
INNER JOIN
twitter_schema.Product P
ON P.product_id = O.product_id;
```

12) Use Case 12: View the items sold by a seller

```
CREATE VIEW twitter_schema.items_sold_by_the_seller AS
SELECT P.product_name as product_name
FROM
(SELECT product_id
FROM twitter_schema.twitter_order_header oh
INNER JOIN twitter_schema.twitter_order_details od
ON oh.t_order_id = od.t_order_id
WHERE seller_id = 1) O
INNER JOIN
twitter_schema.Product P
ON P.product_id = O.product_id;
```

13) Use Case 13: View the top selling product

```
CREATE VIEW twitter_schema.top_selling_product AS
SELECT product_name, ORDER_CNT
FROM
(SELECT product_id, COUNT(oh.t_order_id) ORDER_CNT
FROM twitter_schema.twitter_order_header oh
INNER JOIN twitter_schema.twitter_order_details od
ON oh.t_order_id = od.t_order_id
GROUP BY 1
ORDER BY ORDER_CNT DESC
LIMIT 1) O
INNER JOIN
twitter_schema.Product P
ON P.product_id = O.product_id;
```

14) Use Case 14: View the seller with highest number of 5-star feedbacks

```
CREATE VIEW twitter_schema.seller_with_highest_number_of_5_star_feedbacks AS
SELECT seller_id, COUNT(T_order_id) 5_STAR_TXN_COUNT
FROM twitter_schema.twitter_order_header T
LEFT JOIN
twitter_schema.Feedback F
ON F.feedback_id = T.feedback_id
WHERE feedback_score = 5
GROUP BY 1
ORDER BY 5_STAR_TXN_COUNT DESC
LIMIT 1;
```

15) Use Case 15: What is the average number of transactions per seller for sellers without premium subscription?

```
CREATE VIEW twitter_schema.avg1_nbr_of_trans_per_seller_without_premium_subs AS
SELECT AVG(Order_Count) Avg_Transactions_Per_Seller
FROM
(SELECT T.seller_id as seller_id, COUNT(T_order_id) Order_Count
FROM twitter_schema.twitter_order_header T
INNER JOIN
twitter_schema.Seller S
ON S.seller_id = T.seller_id
WHERE S.premium_flag = 0
GROUP BY 1) D;
```

16) Use Case 16: Which State has the most number of buyers?

```
CREATE VIEW twitter_schema.state_that_has_the_most_number_of_buyers AS
SELECT U.state as State, COUNT(B.buyer_id) Buyer_Count
FROM twitter_schema.University U
LEFT JOIN
twitter_schema.Student S
ON S.university_id = U.university_id
LEFT JOIN
twitter_schema.Buyer B
ON B.student_id = S.student_id
GROUP BY State
ORDER BY Buyer_Count DESC
LIMIT 1;
```

17) Use Case 17: View premium seller with highest total sales amount

```
CREATE VIEW twitter_schema.premium_seller_with_highest_total_sales_amount AS
SELECT oh.seller_id as seller_id, SUM(od.price) as Total_Sales
FROM twitter_schema.twitter_order_header oh
INNER JOIN twitter_schema.twitter_order_details od
ON oh.t_order_id = od.t_order_id
INNER JOIN
twitter_schema.Seller S
ON S.seller_id = oh.seller_id
WHERE premium_flag = 1
GROUP BY seller_id
ORDER BY Total_Sales DESC
LIMIT 1;
```

18) Use Case 18: University wise number of students

```
CREATE VIEW twitter_schema.university_wise_number_of_students AS
SELECT university_name, COUNT(student_id) student_count
FROM twitter_schema.University U
INNER JOIN
twitter_schema.Student S
ON S.university_id = U.university_id
GROUP BY 1;
```

19) Use Case 19: View premium seller with lowest total sales amount

```
CREATE VIEW twitter_schema.premium_seller_with_lowest_total_sales_amount AS
SELECT oh.seller_id as seller_id, SUM(od.price) as Total_Sales
FROM twitter_schema.twitter_order_header oh
INNER JOIN twitter_schema.twitter_order_details od
ON oh.t_order_id = od.t_order_id
INNER JOIN
twitter_schema.Seller S
ON S.seller_id = oh.seller_id
WHERE premium_flag = 1
GROUP BY seller_id
ORDER BY Total_Sales ASC
LIMIT 1;
```

20) Use Case 20: Which State has the least number of buyers?

```
CREATE VIEW twitter_schema.state_that_has_the_least_number_of_buyers AS
SELECT U.state as State, COUNT(B.buyer_id) Buyer_Count
FROM twitter_schema.University U
LEFT JOIN
twitter_schema.Student S
ON S.university_id = U.university_id
LEFT JOIN
twitter_schema.Buyer B
ON B.student_id = S.student_id
GROUP BY State
ORDER BY Buyer_Count ASC
LIMIT 1;
```

SQL FOR INSERT INTO NORMALIZED DATABASE

```
insert into twitter_schema.university (university_id, university_name, state, city) values (1, 'Northeastern University 1','MA1','Boston1');

insert into twitter_schema.university (university_id, university_name, state, city) values (2, 'Northeastern University 2','MA2','Boston2');

insert into twitter_schema.university (university_id, university_name, state, city) values (3, 'Northeastern University 3','MA3','Boston3');

insert into twitter_schema.university (university_id, university_name, state, city) values (4, 'Northeastern University 4','MA4','Boston4');

insert into twitter_schema.university (university_id, university_name, state, city) values (5, 'Northeastern University 5','MA5','Boston5');


insert into twitter_schema.student (student_id, university_id, first_name, last_name) values (1,1,'Tejas 1','Parikh 1');

insert into twitter_schema.student (student_id, university_id, first_name, last_name) values (2,2,'Tejas 2','Parikh 2');

insert into twitter_schema.student (student_id, university_id, first_name, last_name) values (3,3,'Tejas 3','Parikh 3');

insert into twitter_schema.student (student_id, university_id, first_name, last_name) values (4,4,'Tejas 4','Parikh 4');

insert into twitter_schema.student (student_id, university_id, first_name, last_name) values (5,5,'Tejas 5','Parikh 5');


insert into twitter_schema.buyer (buyer_id, student_id) values (1,1);

insert into twitter_schema.buyer (buyer_id, student_id) values (2,2);

insert into twitter_schema.buyer (buyer_id, student_id) values (3,3);
```

```
insert into twitter_schema.seller (seller_id, student_id,premium_flag) values (4,4,1);

insert into twitter_schema.seller (seller_id, student_id,premium_flag) values (5,5,0);
```

```
insert into twitter_schema.tweet ( twitter_handle, tweet_text, profile_image_url,
tweet_date, user_created_at, retweets) values
('abc1','sample_text1','profile_image_url_1','2022-12-12','2022-12-12',21);

insert into twitter_schema.tweet ( twitter_handle, tweet_text, profile_image_url,
tweet_date, user_created_at, retweets) values
('abc2','sample_text2','profile_image_url_2','2022-11-11','2022-11-11',22);

insert into twitter_schema.tweet ( twitter_handle, tweet_text, profile_image_url,
tweet_date, user_created_at, retweets) values
('abc3','sample_text3','profile_image_url_3','2022-10-10','2022-10-10',23);

insert into twitter_schema.tweet ( twitter_handle, tweet_text, profile_image_url,
tweet_date, user_created_at, retweets) values
('abc4','sample_text4','profile_image_url_4','2022-12-11','2022-12-11',24);

insert into twitter_schema.tweet ( twitter_handle, tweet_text, profile_image_url,
tweet_date, user_created_at, retweets) values
('abc5','sample_text5','profile_image_url_5','2022-11-12','2022-11-12',25);
```

```
insert into twitter_schema.tweet_mentions (tweet_id,source_user, target_user) values
(1,'Sam5','Antony1');

insert into twitter_schema.tweet_mentions (tweet_id,source_user, target_user) values
(2,'Sam1','Antony2');

insert into twitter_schema.tweet_mentions (tweet_id,source_user, target_user) values
(3,'Sam2','Antony3');

insert into twitter_schema.tweet_mentions (tweet_id,source_user, target_user) values
(4,'Sam3','Antony4');

insert into twitter_schema.tweet_mentions (tweet_id,source_user, target_user) values
(5,'Sam4','Antony5');
```

```
insert into twitter_schema.tweet_tags (tweet_id,tag, target_user) values
(1,'tag1','Antony1');

insert into twitter_schema.tweet_tags (tweet_id,tag, target_user) values
(2,'tag2','Antony2');

insert into twitter_schema.tweet_tags (tweet_id,tag, target_user) values
(3,'tag3','Antony3');

insert into twitter_schema.tweet_tags (tweet_id,tag, target_user) values
(4,'tag4','Antony4');

insert into twitter_schema.tweet_tags (tweet_id,tag, target_user) values
(5,'tag5','Antony5');


insert into twitter_schema.category (category_id,category_name) values (1,'Furniture');

insert into twitter_schema.category (category_id,category_name) values (2,'Furniture');

insert into twitter_schema.category (category_id,category_name) values (3,'Clothing');

insert into twitter_schema.category (category_id,category_name) values (4,'Furniture');

insert into twitter_schema.category (category_id,category_name) values (5,'Clothing');


insert into twitter_schema.feedback (feedback_id,feedback_score,comments) values
(1,4,'Good Product');

insert into twitter_schema.feedback (feedback_id,feedback_score,comments) values
(2,5,'Good Product');

insert into twitter_schema.feedback (feedback_id,feedback_score,comments) values
(3,2,'Worst Product');

insert into twitter_schema.feedback (feedback_id,feedback_score,comments) values
(4,2,'Worst Product');

insert into twitter_schema.feedback (feedback_id,feedback_score,comments) values
(5,4,'Good Product');
```

```
insert into twitter_schema.product (product_id,product_name,category_id) values
(1,'Table',1);

insert into twitter_schema.product (product_id,product_name,category_id) values
(2,'Chair',2);

insert into twitter_schema.product (product_id,product_name,category_id) values
(3,'Scarf',3);

insert into twitter_schema.product (product_id,product_name,category_id) values
(4,'Cupboard',4);

insert into twitter_schema.product (product_id,product_name,category_id) values
(5,'Shirt',5);


INSERT INTO twitter_schema.twitter_order_header
(t_order_id,buyer_id,seller_id,tweet_id,feedback_id,order_date) values (1,1,4,1,1,'2022-12-
12');

INSERT INTO twitter_schema.twitter_order_header
(t_order_id,buyer_id,seller_id,tweet_id,feedback_id,order_date) values (2,2,4,2,2,'2012-12-
12');

INSERT INTO twitter_schema.twitter_order_header
(t_order_id,buyer_id,seller_id,tweet_id,feedback_id,order_date) values (3,3,5,3,3,'2002-12-
12');

INSERT INTO twitter_schema.twitter_order_header
(t_order_id,buyer_id,seller_id,tweet_id,feedback_id,order_date) values (4,1,4,4,4,'2021-12-
12');

INSERT INTO twitter_schema.twitter_order_header
(t_order_id,buyer_id,seller_id,tweet_id,feedback_id,order_date) values (5,2,5,5,5,'2020-12-
12');
```

```
INSERT INTO
twitter_schema.twitter_order_details(t_order_details_id,t_order_id,product_id,price)
values (1,1,1,200.56);

INSERT INTO
twitter_schema.twitter_order_details(t_order_details_id,t_order_id,product_id,price)
values (2,2,2,100.56);

INSERT INTO
twitter_schema.twitter_order_details(t_order_details_id,t_order_id,product_id,price)
values (3,3,3,60.56);

INSERT INTO
twitter_schema.twitter_order_details(t_order_details_id,t_order_id,product_id,price)
values (4,4,4,4000.56);

INSERT INTO
twitter_schema.twitter_order_details(t_order_details_id,t_order_id,product_id,price)
values (5,5,5,2000.56);
```