Assignment 2: Scraping Twitter

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

Topic: University Marketplace Database

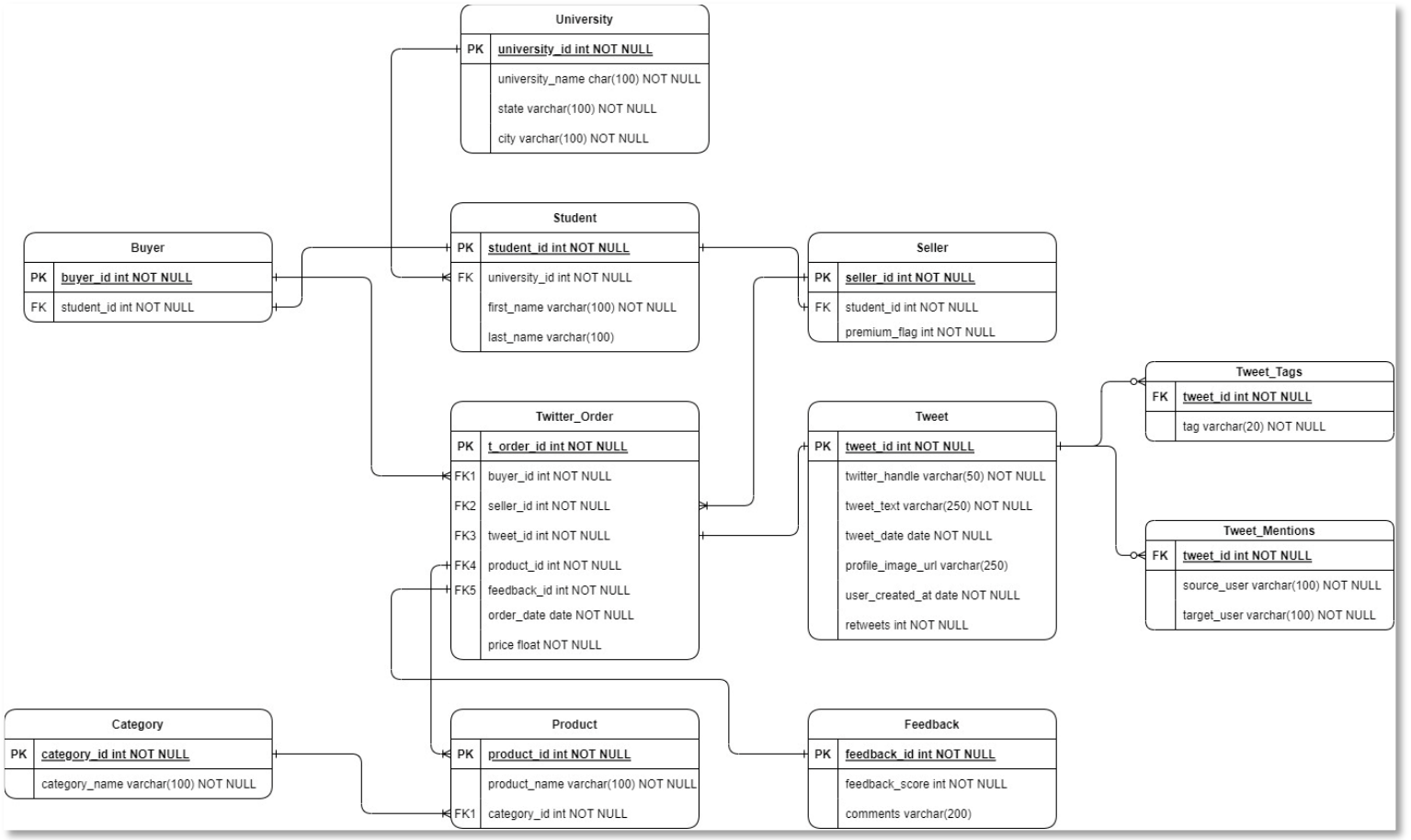
Topic description:

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.

Updated Conceptual Model (Also included as a separate JPG file)



SQL for the physical model that represents the conceptual model:

University Table:

```
CREATE TABLE University (
        university_id INT,
    university_name VARCHAR(100) NOT NULL,
    state VARCHAR(100) NOT NULL,
    city VARCHAR(100) NOT NULL,
    PRIMARY KEY (university_id)
);
```

Student Table:

```
CREATE TABLE Student (
        student_id INT,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100),
    university_id INT NOT NULL,
    PRIMARY KEY (student_id),
    FOREIGN KEY (university_id) REFERENCES University(university_id)
);
```

Seller Table:

```
CREATE TABLE Seller (
        seller_id INT,
    student_id INT,
    premium_flag INT NOT NULL,
    PRIMARY KEY (seller_id),
    FOREIGN KEY (student_id) REFERENCES Student(student_id)
```

);

Buyer Table:

```
CREATE TABLE Buyer (
        buyer_id INT,
    student_id INT,
    PRIMARY KEY (buyer_id),
    FOREIGN KEY (student_id) REFERENCES Student(student_id)
);
```

Tweet Table:

```
CREATE TABLE Tweet (
        tweet_id INT,
    twitter_handle varchar(50) NOT NULL,
    tweet_text varchar(250),
    tweet_date DATE NOT NULL,
    profile_image_url varchar(250),
    user_created_at date NOT NULL,
    retweets int NOT NULL,
    PRIMARY KEY (tweet_id)
);
```

Category Table:

```
CREATE TABLE Category (
        category_id INT,
    category_name varchar(100) NOT NULL,
```

```
    PRIMARY KEY (category_id)
);
```

Product Table:

```
CREATE TABLE Product (
        product_id INT,
    product_name varchar(100) NOT NULL,
    category_id INT NOT NULL,
    PRIMARY KEY (product_id),
    FOREIGN KEY (category_id) REFERENCES Category(category_id)
);
```

Feedback Table:

```
CREATE TABLE Feedback (
        feedback_id INT,
    feedback_score INT NOT NULL,
    comments varchar(200),
    PRIMARY KEY (feedback_id)
);
```

Twitter_Order Table:

```
CREATE TABLE Twitter_Order (
        t_order_id INT,
    buyer_id INT NOT NULL,
    seller_id INT NOT NULL,
    tweet_id INT NOT NULL,
```

```
    product_id INT NOT NULL,
    feedback_id INT NOT NULL,
    order_date DATE NOT NULL,
    price FLOAT NOT NULL,
    PRIMARY KEY (t_order_id),
    FOREIGN KEY (buyer_id) REFERENCES Buyer(buyer_id),
    FOREIGN KEY (seller_id) REFERENCES Seller(seller_id),
    FOREIGN KEY (tweet_id) REFERENCES Tweet(tweet_id),
    FOREIGN KEY (product_id) REFERENCES Product(product_id),
    FOREIGN KEY (feedback_id) REFERENCES Feedback(feedback_id)
);
```

Tweet_Tags Table:

```
CREATE TABLE Tweet_Tags (
        tweet_id INT,
    tag varchar(20) NOT NULL,
    FOREIGN KEY (tweet_id) REFERENCES Tweet(tweet_id)
);
```

Tweet_Mentions Table:

```
CREATE TABLE Tweet_Mentions (
        tweet_id INT,
    source_user varchar(100) NOT NULL,
    target_user varchar(100) NOT NULL,
    FOREIGN KEY (tweet_id) REFERENCES Tweet(tweet_id)
);
```

SQL and Relational Algebra that expresses the queries that we are asked to write

What user posted this tweet?

SQL –

```
SELECT t.twitter_handle, t.tweet_id
FROM twitter_schema.Tweet t
WHERE t.tweet_id = 1;
```

Relational Algebra -

$\Pi_{Twitter\_Schema.twitter\_handle,Twitter\_Schema.tweet\_id}(\sigma_{twitter\_schema.tweet\_id = 1}(Tweet))$

When did the user post this tweet?

SQL –

```
SELECT t.twitter_handle, t.tweet_date, t.tweet_id
FROM twitter_schema.Tweet t
WHERE t.tweet_id = 1;
```

Relational Algebra -

$\Pi_{Twitter\_Schema.twitter\_handle,Twitter\_Schema.tweet\_id,Twitter\_Schema.tweet\_date}(\sigma_{twitter\_schema.tweet\_id = 1}(Tweet))$

What tweets have this user posted in the past 24 hours?

SQL –

```
SELECT t.tweet_id,t.twitter_handle,t.tweet_text,t.tweet_date
FROM twitter_schema.Tweet t
WHERE t.tweet_date >= now() – INTERVAL 1 DAY and twitter_handle = ‘IndiraB38555421’ ;
```

Relational Algebra -

$\Pi_{Twitter\_Schema.twitter\_handle,Twitter\_Schema.tweet\_id,Twitter\_Schema.tweet\_date,\ Twitter\_Schema.tweet\_text}(\sigma_{twitter\_schema.tweet\_date >= now() – INTERVAL\ 1\ DAY\ \wedge\ Twitter\_Schema.twitter\_handle\ = ‘IndiraB38555421’}(Tweet))$

How many tweets have this user posted in the past 24 hours?

SQL –

```
SELECT  t.tweet_id,t.twitter_handle,t.tweet_text,t.tweet_date,count(t.twitter_handle) as ‘Number of Tweets’
FROM twitter_schema.Tweet t
WHERE t.tweet_date >= now() – INTERVAL 1 DAY and t.twitter_handle = ‘IndiraB38555421’ ;
```

Relational Algebra -

$\Pi_{Twitter\_Schema.twitter\_handle,Twitter\_Schema.tweet\_id,Twitter\_Schema.tweet\_date,\ Twitter\_Schema.tweet\_text}$, $\Im_{count\ twitter\_schema.twitter\_handle}$ ($\sigma_{twitter\_schema.tweet\_date >= now() - INTERVAL\ 1\ DAY\ \wedge\ Twitter\_Schema.twitter\_handle = 'IndiraB38555421'}$ (Tweet))

When did this user join Twitter?

SQL –

```
SELECT  t.twitter_handle,t.user_created_at AS 'Twitter_Joining_Date'
FROM twitter_schema.Tweet t
WHERE t.twitter_handle = 'ChrisTh67699326' ;
```

Relational Algebra -

$P_{Twitter\_Schema(twitter\_handle,Twitter\_Joining\_Date)}$ $\Pi_{Twitter\_Schema.twitter\_handle}$ ($\sigma_{twitter\_schema.twitter\_handle = 'ChrisTh67699326'}$ (Tweet))

What keywords/ hashtags are popular?

SQL –

```
SELECT tt.tag AS 'Popular_Tag', count(tt.tag) AS 'Tag_Count'
FROM twitter_schema.tweet t
INNER JOIN twitter_schema.tweet_tags tt
ON  t.tweet_id = tt.tweet_id
Group by tt.tag
ORDER BY 2 desc
LIMIT 3;
```

Relational Algebra -

$P_{Twitter\_Tag(Popular\_Tag,Tag\_Count)}$ $\Pi_{twitter\_tag.tag,\ \Im count\ twitter\_schema.twitter\_tag}$ (Twitter_Schema $\bowtie_{twitter\_schema.tweet\_id = tweet.tweet\_id}$ (Tweet))

What tweets are popular?

SQL –

```
SELECT t.twitter_handle, t.tweet_text, t.tweet_date, t.retweets
FROM twitter_schema.tweet t
ORDER BY t.retweets desc
LIMIT 5 ;
```

Relational Algebra -

$\Pi_{Twitter\_Schema.twitter\_handle,Twitter\_Schema.tweet\_id,\ Twitter\_Schema.tweet\_text,\ Twitter\_Schema.retweets}$($\tau_{Twitter\_Schema.retweets\ DESC}$ (Tweet))

Use Cases of queries that are particular to our domain

1) Use Case 1: Seller tweets to sell an item

Description: Seller tweets on the Twitter platform to sell a used/new item.
Actor: Seller
Precondition: When a seller wants to sell an item on the Twitter platform, he needs to sign up first. Also, since our database is catered to university students, we will verify if the user (buyer or seller) belongs to a university by valid their email address in the registration
Steps-
Actor action: Seller creates an account on Twitter
System Responses: If the seller is successfully registered, he/she should be able to tweet and mention which items he needs to sell.
Post Condition: Seller successfully able to tweet on Twitter

SQL -

INSERT INTO Tweet (twitter_handle, tweet_text, tweet_date, profile_image_url, user_created_at, retweets) VALUES ('minniefaries','I am SELLING MY TABLE FOR FREE FOR UNIVERSITY STUDENTS!!!','2022-12-11', 'https://twitter.com/minniefaries/photo','2021-12-12',5)
INSERT INTO Product (product_name) VALUES ('table')
INSERT INTO Category (category_name) VALUES ('furniture')
INSERT INTO Twitter_order (t_order_id, buyer_id, seller_id, tweet_id, product_id,feedback_id, order_date, price) VALUES (1384,787,91,119,33,1014,'2022-12-12',1000)

Relational Algebra -

Tweet <- Tweet U {('minniefaries','I am SELLING MY TABLE FOR FREE FOR UNIVERSITY STUDENTS!!!','2022-12-11','https://twitter.com/minniefaries/photo','2021-12-12',5)}
Product <- Product U {('table')}
Category <- Category U {('furniture')}
Twitter_order <- Twitter_order U {(1384,787,91,119,33,1014,'2022-12-12',1000)}

2) Use Case 2: View the items that can be purchased by the buyer

Description: The buyer should be able to view all the products that he/she can buy on the platform.

Actor: Buyer

Precondition: When a buyer wants to buy an item on the Twitter platform, he needs to be able to successfully sign up first.

Steps-

Actor action: Buyer can view the item that he can purchase on Twitter.

System Responses: If the buyer is meeting all the preconditions, he should be able to view the items that can be purchased.

Post Condition: Buyer successfully able to view tweets on Twitter.

Alternate Path: The buyer is not able to view products since the system might be down or the product is temporarily unavailable.

Error: If the buyer is not able to view the products, then throw the “Items cannot be viewed during this time" error.

SQL -

```
SELECT p.product_name, c.category_name
FROM Product p
INNER JOIN Category c
ON c.category_id = p.category_id
```

Relational Algebra -

$\Pi_{\text{product.product\_name, category.category name}} (\text{Product} \bowtie_{\text{Product.category\_id = Category.category\_id}} (\text{Category}))$

3) Use Case 3: Which seller from the University sold the most items

Description: For a particular university, we can see which seller sold the most items.

Actor: User (buyer or another seller)

Precondition: The user needs to sign up first in order to view the seller from the university that sold most items.

Steps-

Actor action: A user from a particular university can view the user that sold the most items

System Responses: If the User is successfully registered, he/she should be able to view the seller from the university that sold most items.

Post Condition: The user is successfully able to view the seller who sold most items.

Alternate Path: The user is not able to view sellers which sold most items if the system is down and an error is thrown.

Error: If the user is not able to view the item, then throw the "Items cannot be viewed during this time" error.

SQL –

```
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
        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
```

Relation Algebra –

$\Pi_{\text{Student.first\_name, Student.last\_name, Twitter\_Order. Number\_of\_items\_sold}}($

$\Pi_{\text{twitter\_order.seller\_id}}, p(\Im_{\text{count twitter\_order.product\_id}} P(\text{Number\_of\_items\_sold}))$
$(\text{Twitter\_Order} \bowtie_{\text{Twitter\_Order.seller\_id} = \text{Seller.seller\_id}} (\text{Seller}))$
$\wedge$ $(\text{Seller} \bowtie_{\text{Seller.student\_id} = \text{Student.student\_id}} (\text{Student}))$

4) Use Case 4: View a product below a particular price

Description: The user should be able to view a product below a certain price

Actor: User (Buyer or seller)

Precondition: The user or a buyer needs to sign up first in order to view the product below a particular price.

Steps-

Actor action: A user or buyer from a particular university can view a product below a particular price.

System Responses: If the User or buyer is registered and the system is not down, he/she should be able to view the products below a particular price.

Post Condition: User or buyer successfully able to view the products below a particular price.

Alternate Path: The user or buyer is not able to view the products below a particular price and an error is thrown.

Error: If the user or buyer is not able to view the products below a particular price then throw the “Items cannot be viewed during this time" error.

SQL -

```
SELECT p.product_name
FROM Product p
INNER JOIN Twitter_Order t
ON p.product_id = t.product_id
WHERE p.price < 5000;
```

Relational Algebra -

$\Pi_{Product.product\_name,Twitter\_Order.price}(Product \bowtie_{Product.product\_id = Twitter\_Order.product\_id} \sigma_{twitter\_order.price<5000} (Twitter\_Order))$

5) Use Case 5: Top 3 orders which received the worst feedback

Description: The user should be able to view the top 3 order id which has received the worst feedback.

Actor: User (Buyer or seller)

Precondition: The user needs to sign up first in order to view the top 3 order IDs that received the worst feedback.

Steps-

Actor action: The user selects the top 3 order id that has received the worst feedback.

System Responses: If the system is not down, he/she should be able to view the top 3 order id which has received the worst feedback.

Post Condition: User was successfully able to view the top 3 order IDs which have received the worst feedback.

Alternate Path: The user is not able to view the top 3 order id that has received the worst feedback and an error is thrown.

Error: If the user is not able to view the top 3 order id which has received the worst feedback, then

throw "Items cannot be viewed during this time" error.

SQL -

```
SELECT t.t_order_id
FROM Twitter_Order t
INNER JOIN Feedback f
ON f.feedback_id = t.feedback_id
ORDER BY feedback_score ASC LIMIT 3;
```

Relational Algebra –

$$\Pi_{\text{Twitter\_Order.t\_order\_id}} (\text{Twitter\_Order} \bowtie_{\text{Twitter\_Order. feedback\_id = Feedback. feedback\_id}} \tau_{\text{Feedback.feedback\_score}}(\text{Feedback}))$$

6) Use Case 6: Top 3 orders which have received the positive feedback

Description: The user should be able to view the top 3 order id which has received positive feedback.

Actor: User (Buyer or seller)

Precondition: The user needs to sign up first in order to view the top 3 order IDs that received positive feedback.

Steps-

Actor action: The user selects the top 3 order id that has received positive feedback.

System Responses: If the system is not down, he/she should be able to view the top 3 order id which has received positive feedback.

Post Condition: The user is successfully able to view the top 3 order IDs which have received positive feedback.

Alternate Path: The user is not able to view the top 3 order id that has received positive feedback and an error is thrown.

Error: If the user is not able to view the top 3 order id which has received the positive feedback, then throw the "Items cannot be viewed during this time" error

SQL -

```
SELECT t.t_order_id
FROM Twitter_Order t
INNER JOIN Feedback f
ON f.feedback_id = t.feedback_id
ORDER BY feedback_score DESC LIMIT 3;
```

Relational Algebra –

$\Pi_{\text{Twitter\_Order.t\_order\_id}}$ (Twitter_Order $\bowtie$ $_{\text{Twitter\_Order. feedback\_id = Feedback. feedback\_id}}$ τ $_{\text{Feedback.feedback\_score DESC}}$ (Feedback))

7) Purchase from a tweet

Description: Buyer purchases an item from a tweet

Actor: Buyer/Student

Precondition:
i) At least one product should be listed by any seller
ii) buyer should be enrolled in a university
iii) buyer should be registered on Twitter

Steps-
Actor action: Select the listing and click buy link
System Responses: Order will be placed for the buyer
Post Condition: Buyer will receive the item delivery
Alternate Path: There are no product listed by any seller
Error: No products available

SQL-

INSERT INTO Twitter_Order(t_order_id, buyer_id, seller_id, tweet_id, product_id,feedback_id, order_date, price)
VALUES (1384,787,91,119,33,1014,"2022-06-07",50);

INSERT INTO Tweet (twitter_handle, tweet_text, tweet_date, profile_image_url, user_created_at, retweets)
VALUES ('minniefaries','I am SELLING MY TABLE FOR FREE FOR UNIVERSITY STUDENTS!!!','2022-12-11','https://twitter.com/minniefaries/photo','2021-12-12',5)

Relational Algebra -

Twitter_Order <- Twitter_Order U {(1384,787,91,119,33,1014,"2022-06-07",50)}

Tweet <- Tweet U {('minniefaries','I am SELLING MY TABLE FOR FREE FOR UNIVERSITY STUDENTS!!!','2022-12-11','https://twitter.com/minniefaries/photo','2021-12-12',5)}

8) View the product with highest price.

Description: Buyer views the product with the highest price
Actor: Buyer/Student
Precondition:
i) At least one product should be listed by any seller
ii) buyer should be enrolled in a university
iii) buyer should be registered on Twitter
Steps-
Actor action: Buyer views the product with highest price
System Responses: the product with highest price is displayed
Post Condition: buyer is successfully able to view the product with highest price
Alternate Path: There are no product listed by any seller
Error: No products available

SQL-

```
SELECT product_name, Price
FROM
(SELECT product_id, MAX(price) Price
FROM Twitter_Order
GROUP BY 1
ORDER BY Price Desc
LIMIT 1) O
INNER JOIN
Product P
ON P.product_id = O.product_id;
```

Relational Algebra -

$\Pi_{(product\_name, PRICE)}$
$(\rho_{(product\_id, PRICE)\ product\_id}\ \Im_{MAX\ price}\ \tau_{PRICE}(Twitter\_Order)) \bowtie Product.product\_id = Twitter\_Order.product\_id\ (Product)$

9) View the items sold by a seller

Description: Seller views the items sold by him/her
Actor: Seller/Student
Precondition: Seller should be enrolled in a university
Steps-
Actor action: Seller requests to view the history of selling items
System Responses: Displays all the details about sales made by the seller
Post Condition: Seller is successfully able to view the sales made by him/her
Alternate Path: There are no sales made by a seller
Error: No history of sales available

SQL-

```
SELECT P.product_name as product_name
FROM
(SELECT product_id
FROM Twitter_Order
WHERE seller_id = 101) O
INNER JOIN
Product P
ON P.product_id = O.product_id;
```

Relational Algebra -

$\Pi_{product\_name}(\sigma_{seller\_id\ =\ 101}(Twitter\_Order) \bowtie Product.product\_id = Twitter\_Order.product\_id\ (Product))$

10) View the top selling product

Description: Buyer views the top selling product
Actor: Buyer/Student
Precondition: There should be some products sold
Steps-
Actor action: Buyer requests to view the top selling product
System Responses: Displays the top selling product with highest selling frequency
Post Condition: Buyer is successfully able to view the top selling product
Alternate Path: There are no products sold
Error: No product sales available

SQL-

```
SELECT product_name, ORDER_CNT
FROM
(SELECT product_id, COUNT(t_order_id) ORDER_CNT
FROM Twitter_Order
GROUP BY 1
ORDER BY ORDER_CNT DISC
LIMIT 1) O
INNER JOIN
Product P
ON P.product_id = O.product_id;
```

Relational Algebra -

$\Pi_{(product\_name,\ ORDER\_CNT)}$
$(\rho_{(product\_id,\ ORDER\_CNT)\ product\_id}\ \Im_{\ COUNT\ t\_order\_id}\ \tau_{ORDER\_CNT}(Twitter\_Order)\ ) \bowtie$
Product.product_id = Twitter_Order.product_id (Product)

11) Enrol student in a university.

Description: A student will be enrolled in a university
Actor: Student
Precondition: Student must have a university email address
Steps-
Actor action: Student requests for enrolment
System Responses: If the university ID and student information is correct, student is enrolled to the university
Post Condition: Student is successfully enrolled to the respective university
Alternate Path: The student information is invalid or university email ID is invalid so system throws an error
Error: The student information is invalid or university email ID is invalid

SQL-

```
INSERT INTO Student (student_id, university_id, first_name, last_name)
VALUES (363,208,'Daniel','Brown');

INSERT INTO Student (student_id, university_id, first_name, last_name)
VALUES (751,41,'Richard','Bugrara');
```

Relational Algebra –

Student <- Student U {(363,208,'Daniel','Brown')}

Student <- Student U {(751,41,'Richard','Bugrara')}