Bits & Bots Final Project

July.30.2021

—

Nicholas S. Peitong Z. Minhye K. Taiyi C.
Team II - CSE_3241

Part I -- The Final Report

Section 1 -- Database Description

I. Introduction and Project Summary

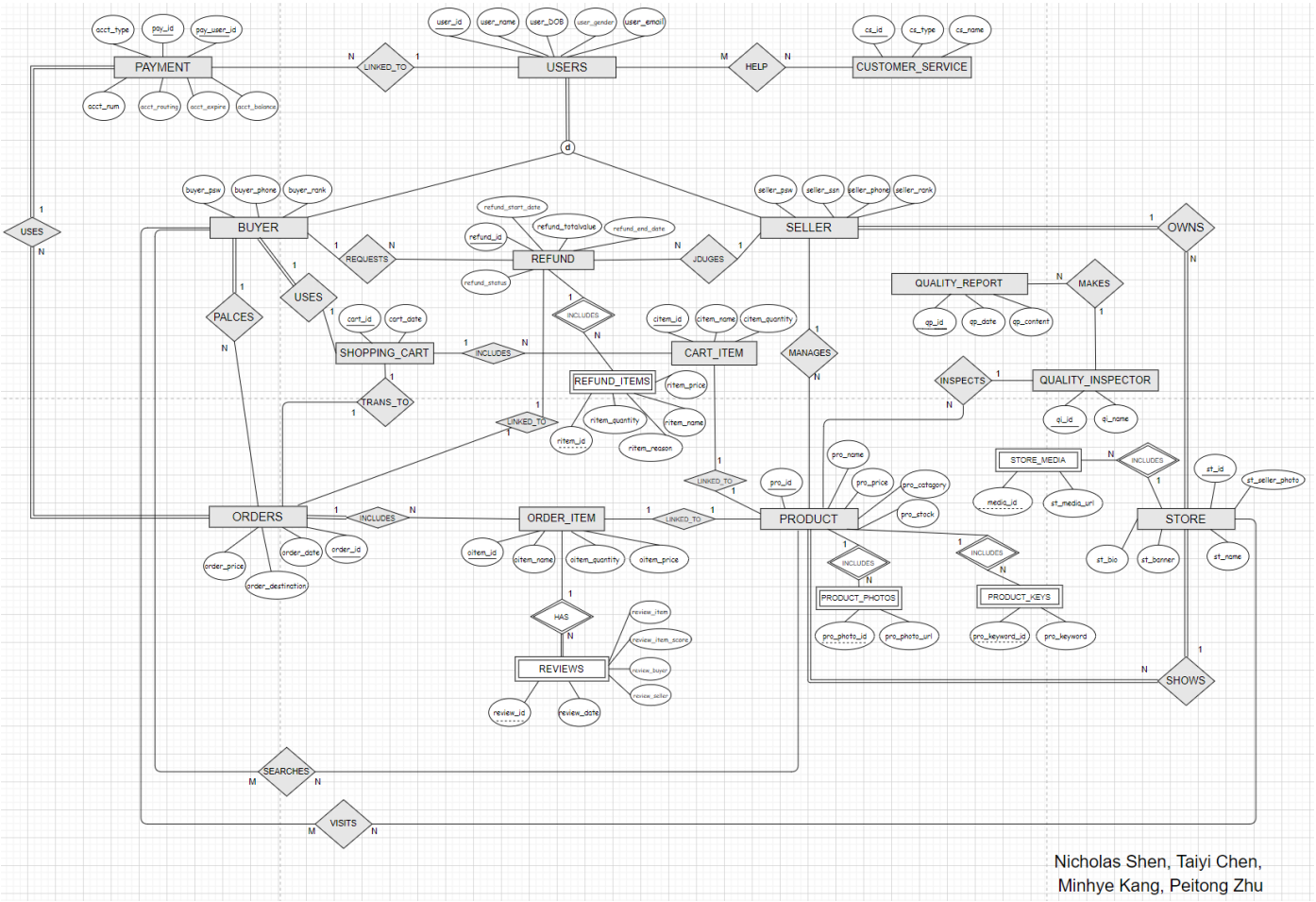
For more details about our team, please see our orientation video here: https://www.youtube.com/watch?v=Ja1GzJDZhls&ab_channel=NickS

Our team is CSE 3241 Team 2. The team members are: Nicholas Shen, Taiyi Chen, Minhye Kang, Peitong Zhu. Our team is employed by DB 4Ever, a consulting company with clients worldwide. We have been assigned to help Ms. Yotta Bietz set up a database for her latest entrepreneurial enterprise, BITS & BOTS, which is an online marketplace for the maker community. Our team is required to make an information management system and database to support virtual inventory, buyer/seller accounts, sales, feedback operations, and etc.

Our project is a database that supports the usual operations of an online store. However, this store only sells non-physical items, such as pdf books and source code. Buyers can search the products they want and add them to a shopping cart. They can make purchases for different items from different shops in one order. Sellers can list their products online, manage their inventory, and view feedback from the buyers. There are many other functions that our database can do, details will be shown later. Extra features of our product are that users can ask the customer service for help, and there is

a quality inspection system, which can report the poor quality of a certain product so that it can be updated or removed from the market.

II. ER Diagram

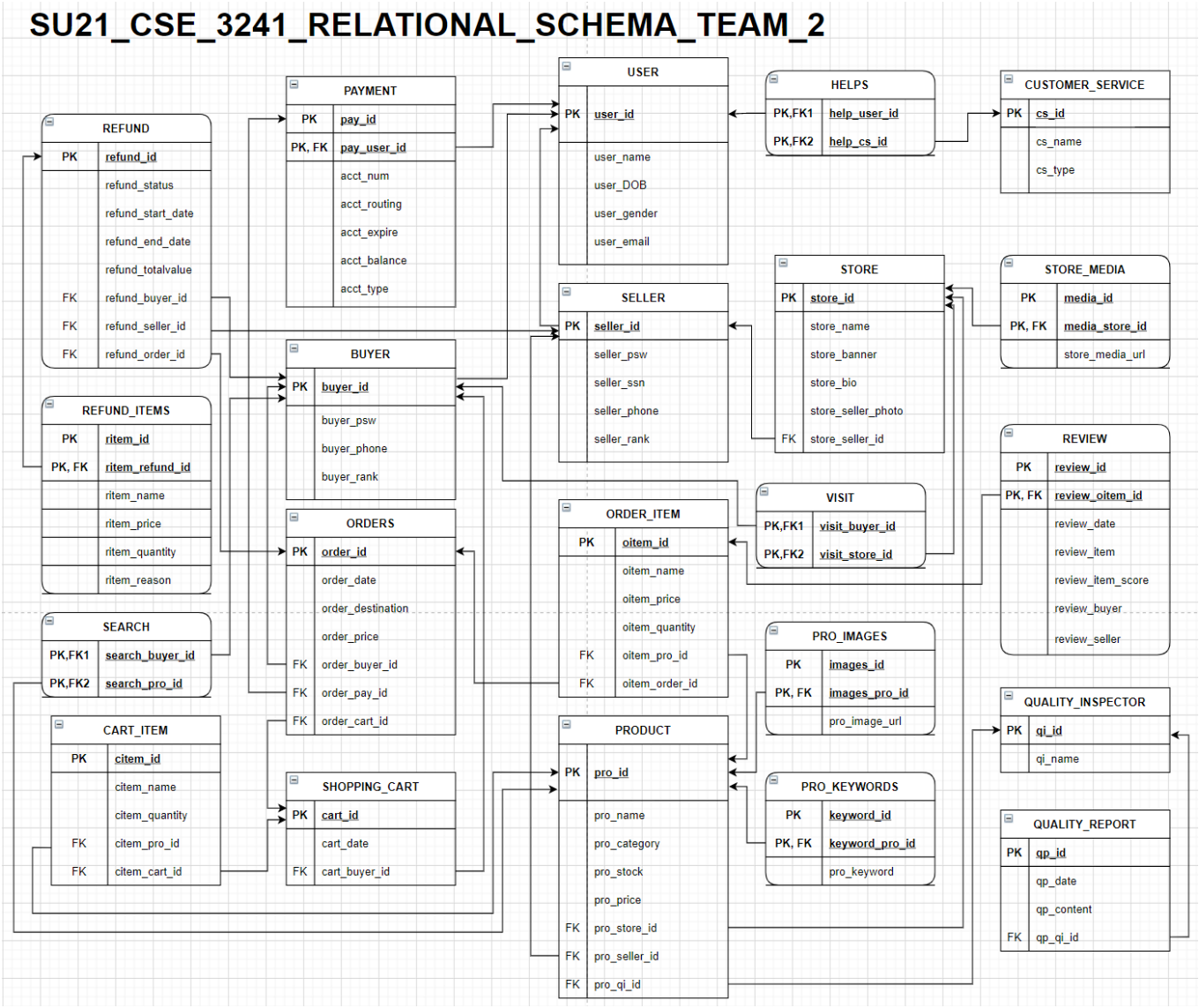


For a better view, please see .png file in ATTACHMENT or use the draw.io link below:
https://drive.google.com/file/d/1agOwf0lWnXxBnASTyd1IJCcxB6T7O0MS/view?usp=sharing

ERD Explanation:

All the attributes are shown in the diagram. Users can ask customer service for help. Users are linked to payment systems, which pay and receive money. Users are separated into buyer and seller. Buyers can search for products and visit stores. Buyers can add items to the shopping carts and place orders. Buyers can also request a refund. Sellers can judge whether those refunding requests are reasonable. Of course, refund information includes refund items as well. They can also manage products and their stores. The shopping carts include ordered items and can be transferred into real orders. Orders are linked to refund and include order items connecting to products. Ordered items have reviews made by buyers. Finally, all products have quality inspectors who make quality reports on specific items.

III. Relational Schema



For a better view, please see .png file in ATTACHMENT or use the draw.io link below:
https://drive.google.com/file/d/1nV6ByEymFAYSMubiAAUTvHnDHgevy4fV/view?usp=sharing

Schema Explanation:

Step 1: Mapping each regular entity type into a relation.
The regular entities are payments, users, customer_service, buyer, orders, refund, shopping_cart, order_items, product, seller, store, quailty_inspector, and quality_report. Each of their simple attributes maps to an attribute of the relation. All derived attributes, such as price, are not necessary to be represented. The primary key of the entity type maps to the primary key of the relation.

Step 2: Mapping weak entities.
The weak entities in our ERD are refund_items and reviews. Our team created a new relation and included all simple attributes of the entity type as attributes of the new relation. And these relations include a primary key attribute of the owner as a foreign key.

Step3: Mapping of binary 1:N relationship types.
The 1:N relationships in our EDR are: “uses” between payments and orders, “places” between buyer and orders, “uses” between buyer and shopping_cart, “requests” between buyer and refund, and so on. We add the key attributes from the “1” side as a foreign key to the relation on the other side. No new relation is added.

Step 4: Mapping of binary 1:1 relationship types.
The 1:1 relationships in our EDR are “uses” between buyer and shooping_cart, “trans_to” between shooping_cart and orders, “linked_to” between cart_items and product. We add the key attributes from the total participation side as a foreign key to the relation on the partial participation side.

Step 5: Mapping of binary M:N relationship types.
For each binary M:N relationship in our ERD, we created a new relation, including the primary key of participating entity types as foreign key attributes in the new relation. All simple attributes of M:N relationship types are included as well. For example, in our relational schema, SEARCH is such a new relation. Its primary keys, as well as foreign keys, are userID and proID.

Since our relationship is in 3NF, there are no multivalued attributes. So, the description above is all the steps we take in the process.

IV. Relational Algebra

3. Given your relational schema, provide the relational algebra to perform the following queries. If your schema cannot provide answers to these queries, revise your ER Model and your relational schema to contain/supply the appropriate information for these queries:

a. Find the titles of all IP Items that cost less than $10 and the name of the stores selling those

π pro_name (σ pro_price < 10 (PRODUCT)) $\bowtie$ pro_id= pro_id STORE

b. Give all the titles and their dates of purchase made by given Buyer (you choose how to designate the buyer)

T1 $\leftarrow$ σ user_id = "selected user" (ORDER)

T2 $\leftarrow$ T1 $\bowtie$ order_id = order_id (ORDER_ITEM)

T3 $\leftarrow$ T2 $\bowtie$ pro_id = pro_id (PRODUCT)

π pro_name, order_date(T3)

c. List all the buyers who purchased an IP Item from a given store and the names of the IP Items they purchased.

T1 $\leftarrow$ σ store = "selected store" (store $\bowtie$ pro_id =pro_id(Product))
T2 $\leftarrow$ T1 $\bowtie$ pro_id = pro_id (ORDER_ITEM)
T3 $\leftarrow$ T2 $\bowtie$ order_id = order_id (order)
T4 $\leftarrow$ T3 $\bowtie$ user_id = user_id (buyer)
T5 $\leftarrow$ T4 $\bowtie$ user_id = user_id (user)
Result $\leftarrow$ π user_name, user_id, pro_name (T5)

d. Find the buyer who has purchased the most IP Items and the total number of IP Items they have purchased.

Using result above
T1 (user_id, numItem) $\leftarrow$ user_id $\Im$ COUNT pro_name (result above)
T2 (user_id, countItem) $\leftarrow$ user_id $\Im$ MAXIMUM numItem (T1)
π name, countItem (T2 $\bowtie$ user_id = user_id (user))

e. Create a list of stores who currently offer 5 or less IP Items for sale

T1 ← STORE ⋈ pro_id = pro_id (PRODUCT)
T2(ID,count) ←store_id ℑ COUNT pro_name
π ID (***σ*** count <=5 (T2))

f. Find the highest selling item, total number of units of that item sold, total dollar sales for that item, and the store/seller who sells it.

T1(ID, count)← pro_id ℑ COUNT pro_id (ORDER_ITEMS)
T2(ID, max) ← ID ℑ MAXIMUM count(T1)
T3 ← T2 ⋈ ID=ProID (PRODUCT)
T4 ← T3 ⋈ userID = userID (SELLER)
π ID,max,userID (T4)

g. Create a list of all payment types accepted, number of times each of them was used, and total amount charged to that type of payment.

T1 ← PAYMENTS ⋈ pay_id = pay_id (ORDER)
T2(acct_type, total) ← acct_type ℑ SUM orderPrice (T1)
T3(acct_type, timesCount) ← acct_type ℑ COUNT acct_type (T1)
T4 ← T2 ⋈ acct_type = acct_type (T3)
RESULT ← **π** acct_type, timesCount, total (T4)

h. Retrieve the name and contact info of the customer who has the highest karma point balance.

T1(ID, max) ← pay_id ℑ MAXIMUM K_acct (PAYMENTS)
T2 ← T1 ⋈ user_id = userID (USER)
π user_name, user_email (T2)

i. Create a list of top 10 rated IP items and the stores selling those.
Adding rating attribute to review:
To display top 1:

T1←Product ⋈ pro_id= pro_id (ORDER_ITEMS)
T2← T1 ⋈ item_id = item_id (Reviews)
T3(pro_id,ave) ← pro_id ℑ AVERAGE rating (T2)
T4 (pro_id,ave) ← pro_id ℑ MAXIMUM ave (T3)
T5 ← T4 ⋈ pro_id = pro_id (STORE)
RESULT ← **π** store_name, pro_title, ave (T5)

4. Three additional interesting queries in plain English and also relational algebra. Each of your queries should include at least one of these:

a. outer joins

Show the title and price of each IP in the shopping cart.

T1 ← PRODUCT ⟕pro_id = pro_id (CART_ITEMS)

T2 ← T1 ⟕ cart_id=cart_id (SHOPPING_CART)

π pro_name, pro_price (T2)

b. aggregate function

List the average cost of IPs in all the stores.

T1 ← STORE ⋈ pro_id = pro_id(PRODUCT)

Result (store_id, average) ← store_id ℑ AVERAGE pro_price (T1)

c. "extra" entities from CP01

Count the number of times customer service was called.

Result (cs_id, num) ← cs_id ℑ COUNT cs_id (CUSTOMER_SERVICE)

V. Database Normalization

PAYMENT is in 3NF. AcctNum, AcctBalance, paymentType and userID are dependent on payID.
ORDERS is in 3NF. orderDate, orderDestination, orderPrice, payID, userID, cartID are dependent on orderID.
SHOPPING_CART is in 3NF. cartDate and userID are dependent on cartID.
ORDER_REFUND is in 3NF. refundDate, refundValue, refundReason, userID, and orderID are dependent on refundID.
USER is in 3NF. userName, userDOB, userGender, userEmail, userPhone are dependent on userID.
BUYER is in 3NF. buyerPassword and buyerCredit are dependent on buyerID.
VISIT is in 3NF. There are no dependencies in this relation.
STORE_URL is in 3NF. storeURL is dependent in storeID.
REVIEW is in 3NF. reviewDate, reviewProduct, reviewBuyer, reviewSeller, itemRating, itemID is dependent on reviewID.
SELLER is in 3NF. sellerPassword and sellerRank are dependent on userID.
STORE is in 3NF. storeName, storeBanner, storeBio, storePhoto, userID are dependent on storeID.
ORDER_ITEM is in 3NF. oitemName, oitemPrice, oitemQuantioty, peoID, orderID are dependent on itemID.
HELPS is in 3NF. There are no dependencies in this relation.
PRO_KEYWORD is in 3NF. proKeyword is dependent on proID.
PRO_PHOTO is in 3NF. proPhotos is dependent on proID.
CUSTOMER_SERVICE is in 3NF. csName and csType are dependent on csID.
SEARCH is in 3NF. There are no dependencies in this relation.
PRODUCT is in 3NF. proName, proPrice, proCategory, stockQuantity, userID, storeID are dependent on proID.
CART_ITEM is in 3NF. citemName, citemQuantity, proID, cartID are dependent on citemID.
INSPECT is in 3NF. There are no dependencies in this relation.
QUALITY_INSPECTOR is in 3NF. qiName is dependent on qiID.
QUALITY_REPORT is in 3NF. reportDate, reportContent, and qiID are dependent on reportID.

Section 2 -- User Manual

S2.A Table description including table functions, keys, constraints, and data types.

USER	USER has the information of both sellers and buyers, any person uses the system. **user_id** int NOT NULL, user_name varchar(50) NOT NULL, user_DOB date NOT NULL, user_gender char(1) NOT NULL DEFAULT 'U', -- M-Male, F-Female, U-Prefer NOT to say user_email varchar(50) NOT NULL UNIQUE
HELPS	HELPS connects the customer to the specific type of the help they requested. **help_user_id** int NOT NULL, **help_cs_id** int NOT NULL, FOREIGN KEY (help_user_id) REFERENCES USER (user_id), FOREIGN KEY (help_cs_id) REFERENCES CUSTOMER_SERVICE (cs_id)
CUSTOMER_SERVICE	CUSTOMER_SERVICE helps users deal with any problems while using the service. **cs_id** int NOT NULL, cs_name varchar(50) NOT NULL, cs_type varchar(50)
PAYMENT	PAYMENT includes the information of the payment that the user could revise. **pay_id** tinyint NOT NULL, **pay_user_id** int NOT NULL, acct_num varchar(50) NOT NULL UNIQUE, acct_routing varchar(50) DEFAULT NULL, acct_expire date DEFAULT NULL, acct_balance int, acct_type varchar(50) NOT NULL DEFAULT 'KarmaPoints',

	FOREIGN KEY (pay_user_id) REFERENCES USER(user_id)
BUYER	BUYER only has the information of the buyer from the USER. **buyer_id** int NOT NULL, buyer_psw varchar(50) NOT NULL, buyer_phone varchar(50), buyer_rank char(1) NOT NULL DEFAULT 'E', /*A to E*/ FOREIGN KEY (buyer_id) REFERENCES USER(user_id)
SHOPPING_CART	SHOPPING_CART includes the date and who has saved the product. **cart_id** int NOT NULL, cart_date date NOT NULL, cart_buyer_id int NOT NULL, FOREIGN KEY (cart_buyer_id) REFERENCES BUYER (buyer_id)
CART_ITEM	CART_ITEM has specific information of the product that the buyer has saved to the shopping cart. **citem_id** int NOT NULL, citem_name varchar(50) NOT NULL, citem_quantity int NOT NULL, citem_pro_id int NOT NULL, citem_cart_id int NOT NULL, FOREIGN KEY (citem_pro_id) REFERENCES PRODUCT (pro_id), FOREIGN KEY (citem_cart_id) REFERENCES SHOPPING_CART (cart_id)
ORDERS	ORDER includes the overview of the order that the buyer has placed, which include how he paid and connected from the SHOPPING_CART. **order_id** int NOT NULL, order_date DATE NOT NULL, order_destination varchar(50) NOT NULL, order_price decimal(6,2) NOT NULL, order_buyer_id int NOT NULL, order_pay_id tinyint NOT NULL, order_cart_id int NOT NULL,

	FOREIGN KEY (order_buyer_id) REFERENCES BUYER (buyer_id), FOREIGN KEY (order_pay_id) REFERENCES PAYMENT_METHOD (pay_id), FOREIGN KEY (order_cart_id) REFERENCES SHOPPING_CART (cart_id)
ORDER_ITEM	ORDER_ITEM has details of the product from the ORDERS that the buyer has placed. **oitem_id** int NOT NULL, oitem_name varchar(50) NOT NULL, oitem_price int NOT NULL, oitem_quantity int NOT NULL, oitem_pro_id int NOT NULL, oitem_order_id int NOT NULL, FOREIGN KEY (oitem_pro_id) REFERENCES PRODUCT (pro_id), FOREIGN KEY (oitem_order_id) REFERENCES ORDERS (order_id)
REVIEWS	REVIEW has the rating for the product from the BUYER. **review_id** int NOT NULL, **review_oitem_id** int NOT NULL, review_date date NOT NULL, review_item varchar(255), review_item_score tinyint, /* rating range: 0 ~ 100 */ review_buyer varchar(255), review_seller varchar(255), FOREIGN KEY (review_oitem_id) REFERENCES ORDER_ITEM (oitem_id) ON DELETE CASCADE
REFUND	REFUND includes both information of buyer to see who has returned the product and seller who sells the product that will be requested to return. **refund_id** int NOT NULL, refund_status varchar(50) NOT NULL, refund_start_date date NOT NULL, refund_finish_date date NOT NULL, refund_value int NOT NULL, refund_buyer_id int NOT NULL, refund_seller_id int NOT NULL,

	refund_order_id int NOT NULL, FOREIGN KEY (refund_seller_id) REFERENCES SELLER (seller_id), FOREIGN KEY (refund_buyer_id) REFERENCES BUYER (buyer_id), FOREIGN KEY (refund_order_id) REFERENCES ORDERS (order_id)
REFUND_ITEMS	REFUND_ITEMS has details of information of the product that has been requested to return. **ritem_id** int NOT NULL, **ritem_refund_id** int NOT NULL, ritem_name varchar(50) NOT NULL, ritem_price int NOT NULL, ritem_quantity int NOT NULL, ritem_reason varchar(255) NOT NULL, FOREIGN KEY (ritem_refund_id) REFERENCES REFUND (refund_id) ON DELETE CASCADE
SEARCH	SEARCH connects to the PRODUCT or BUYER for detail of the product and buyer when the product id or buyer id has been entered. **search_buyer_id** int NOT NULL, **search_pro_id** int NOT NULL, FOREIGN KEY (search_buyer_id) REFERENCES BUYER (buyer_id), FOREIGN KEY (search_pro_id) REFERENCES PRODUCT (pro_id)
SELLER	SELLER only has the information of the buyer from the USER. **seller_id** int NOT NULL, seller_psw varchar(50) NOT NULL, seller_ssn varchar(9) NOT NULL UNIQUE, seller_phone varchar(50) NOT NULL, seller_rank char(1) NOT NULL DEFAULT 'E', /*A to E*/ FOREIGN KEY (seller_id) REFERENCES USER(user_id)
STORE	STORE has the information of the store and connects with SELLER since it is run by a seller.

	store_id int NOT NULL, store_name varchar(50) NOT NULL UNIQUE, store_banner varchar(255) DEFAULT 'No banner', store_bio varchar(255) DEFAULT 'No bio', store_seller_photo varchar(255), store_seller_id int NOT NULL, FOREIGN KEY (store_seller_id) REFERENCES SELLER (seller_id)
STORE_MEDIA	STORE_MEDIA is for advertising the store. **media_id** tinyint NOT NULL, **media_store_id** int NOT NULL, store_media_url varchar(255), FOREIGN KEY (media_store_id) REFERENCES STORE (store_id) ON DELETE CASCADE
PRODUCT	PRODUCT has information of the product that seller sells and sells from the store. **pro_id** int NOT NULL, pro_name varchar(50) NOT NULL, pro_category varchar(50), pro_stock int NOT NULL, pro_price decimal(6,2) NOT NULL, pro_store_id int NOT NULL, pro_seller_id int NOT NULL, pro_qi_id int NOT NULL, FOREIGN KEY (pro_store_id) REFERENCES STORE (store_id), FOREIGN KEY (pro_seller_id) REFERENCES SELLER (seller_id), FOREIGN KEY (pro_qi_id) REFERENCES QUALITY_INSPECTOR (qi_id)
PRO_IMAGES	PRO_IMAGES has the integers that can be converted to product image. **images_id** tinyint NOT NULL, **images_pro_id** int NOT NULL, pro_image_url varchar(255) NOT NULL, FOREIGN KEY (images_pro_id) REFERENCES PRODUCT (pro_id) ON DELETE CASCADE

PRO_KEYWORDS	PRO_KEYWORDS **keyword_id** tinyint NOT NULL, **keyword_pro_id** int NOT NULL, pro_keyword varchar(50), FOREIGN KEY (keyword_pro_id) REFERENCES PRODUCT (pro_id) ON DELETE CASCADE
VISIT	VISIT is to see who has visited what store. **visit_buyer_id** int NOT NULL, **visit_store_id** int NOT NULL, FOREIGN KEY (visit_buyer_id) REFERENCES BUYER (buyer_id), FOREIGN KEY (visit_store_id) REFERENCES STORE (store_id)
QULITY_INSPECTOR	QUALITY_INSPECTOR can inspect the quality and validity of a product. **qi_id** int NOT NULL, qi_name varchar(50) NOT NULL
QUALITY_REPORT	QUALITY_REPORT shows the summary of products' quality in detail. **qp_id** int NOT NULL, qp_date date, qp_content varchar(1024) NOT NULL, qp_qi_id int NOT NULL, FOREIGN KEY (qp_qi_id) REFERENCES QUALITY_INSPECTOR (qi_id)

S2.B A catalog of supplied SQL Queries With explanations and sample output for each.

SimpleQueries.sql

3.a. Create a list of items under a certain price with stores selling those. In the sample, we are finding all items under $10.

```
SELECT `product`.pro_name AS prudcut_name, `product`.pro_price
AS product_price, `store`.store_name
FROM `product`, `store`
WHERE `store`.store_id = `product`.pro_store_id AND
`product`.pro_price < 10;
```

prudcut_name	product_price	store_name
Xon Picture 1	8.88	Clion Digital
David Copperfield	9.99	Apple PDF
Fake Digital Book 1	9.99	Banana PDF
Xon Picture 3	7.88	Clion Digital
Xon Picture 4	1.88	Clion Digital
Xon Picture 5	0.88	Clion Digital
Xon Picture 6	2.88	Clion Digital
Xon Picture 7	8.88	Clion Digital
Xon Picture 8	8.88	Clion Digital
Xon Picture 9	5.88	Clion Digital
Xon Picture 10	8.88	Clion Digital

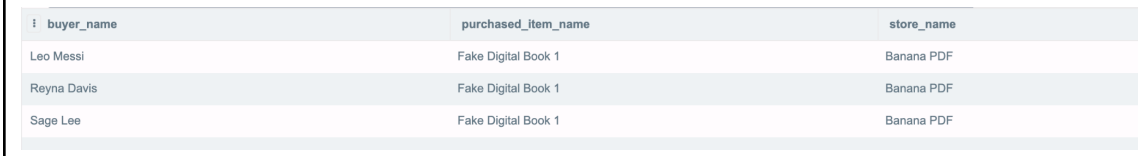
3.b List all past purchased items of users who were born after 1985 with their order date.

```
SELECT `order_item`.oitem_name AS purchased_item,
`orders`.order_date AS purchased_date, `user`.user_name AS
buyer_name, `user`.user_DOB AS buyer_birthdate
FROM `orders`, `order_item`, `user`
WHERE `orders`.order_id = `order_item`.oitem_order_id AND
`orders`.order_buyer_id = `user`.user_id AND `user`.user_DOB
> '1985-01-01';
```

purchased_item	purchased_date	buyer_name	buyer_birthdate
PDF Editor V3.0	2021-07-01	Leo Messi	1987-06-24
Xon Picture 1	2021-07-01	Leo Messi	1987-06-24
David Copperfield	2021-07-02	Leo Messi	1987-06-24
Fake Digital Book 1	2021-07-02	Leo Messi	1987-06-24
Fake Digital Book 6	2021-07-03	Leo Messi	1987-06-24
How to cook	2021-07-06	Cristiano Ronaldo	1985-02-05
Star War III	2021-07-07	Cristiano Ronaldo	1985-02-05
Jave Code for the best calculator	2021-07-07	Cristiano Ronaldo	1985-02-05

3.c. List all the buyers who purchased an IP Item from a given store(*store_id=800052*) and the names of the IP Items they purchased.

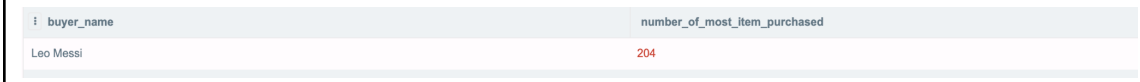
```
SELECT `user`.user_name AS buyer_name, `order_item`.oitem_name
AS purchased_item_name, `store`.store_name
FROM `user`, `buyer`, `orders`, `order_item`, `product`,
`store`
WHERE `store`.store_id = 800052 AND
`order_item`.`oitem_pro_id` = `product`.pro_id AND
`product`.pro_store_id = `store`.store_id
AND `order_item`.`oitem_order_id` = `orders`.order_id AND
`orders`.order_buyer_id = `buyer`.buyer_id AND
`buyer`.buyer_id = `user`.user_id;
```



buyer_name	purchased_item_name	store_name
Leo Messi	Fake Digital Book 1	Banana PDF
Reyna Davis	Fake Digital Book 1	Banana PDF
Sage Lee	Fake Digital Book 1	Banana PDF

3.d. Find the buyer who has purchased the most IP Items and the total number of IP Items they have purchased.

```
SELECT user_name AS buyer_name, MAX(`sub`.total_items) AS
number_of_most_item_purchased
FROM(
     SELECT `user`.user_name,
SUM(`order_item`.oitem_quantity) AS total_items
     FROM `user`, `buyer`, `orders`, `order_item`
     WHERE `order_item`.`oitem_order_id` = `orders`.order_id
AND `orders`.order_buyer_id = `buyer`.buyer_id AND
`buyer`.buyer_id = `user`.user_id
     GROUP BY `user`.user_id
) AS sub;
```



buyer_name	number_of_most_item_purchased
Leo Messi	204

3.e. List the stores that offer more than or less than a certain number of items. In the sample, we list stores with less than 5 items.

```
SELECT store_id, store_name, number_of_products
FROM(
      SELECT `store`.store_id, `store`.store_name,
COUNT(`product`.pro_id) AS number_of_products
      FROM `store`, `product`
      WHERE `store`.store_id = `product`.pro_store_id
      GROUP BY `store`.store_id
) AS sub
WHERE number_of_products < 5;
```

store_id	store_name	number_of_products
800000	Study Online	1
800011	Clion Books	2
800020	Amex Card	1
800030	Wolf PDF	2
800040	Video games Store	1
800050	Aqima	1
800051	Apple PDF	2
800052	Banana PDF	1
800053	Pear PDF	1
800054	Orange PDF	1
800060	The Best Code	1
800070	Digital Gamestop	1
800080	Online Bookstore	1
800090	Slience Online Bookstore	1

3.f. Find the highest selling items, the total number of units of that item sold, total dollar sales for that item, and the store/seller who sells it.

```
SELECT oitem_name AS most_selling_product,
MAX(total_item_sold) AS total_sold_unit, total_sold_price,
store_name, user_name AS seller_name
FROM(
      SELECT oitem_id, oitem_name, oitem_pro_id,
SUM(oitem_quantity) AS total_item_sold, SUM(selling_price) AS
total_sold_price, store_name, user_name
      FROM(
          SELECT `order_item`.oitem_id,
`order_item`.oitem_pro_id, `order_item`.oitem_name,
`order_item`.oitem_quantity, `order_item`.oitem_price,
`order_item`.oitem_quantity*`order_item`.oitem_price AS
selling_price, `store`.store_name, `user`.user_name
          FROM `order_item`,`product`, `store`, `seller`,
`user`
          WHERE `order_item`.`oitem_pro_id` = `product`.pro_id
AND `product`.pro_store_id = `store`.store_id AND
`product`.pro_seller_id = `seller`.seller_id AND
`seller`.seller_id = `user`.user_id
      ) AS sub
      GROUP BY oitem_pro_id
) AS sub2;
```

most_selling_product	total_sold_unit	total_sold_price	store_name	seller_name
PDF Editor V3.0	200	2000	Wolf PDF	Serena Williams

3.g. Create a list of all payment types accepted, the number of times each of them was used, and the total amount charged to that type of payment.

```
SELECT acct_type AS payment_types, COUNT(acct_type) AS
used_times, ROUND(SUM(order_price),2) AS total_charges
FROM(
      SELECT `orders`.order_id, `payment_method`.acct_type,
`orders`.order_price
      FROM `payment_method`, `orders`, `user`
      WHERE `orders`.order_pay_id = `payment_method`.pay_id
AND `payment_method`.pay_user_id = `user`.user_id AND
`user`.user_id = `orders`.order_buyer_id
      GROUP BY `orders`.order_id
) AS sub
GROUP BY acct_type;
```

payment_types	used_times	total_charges
Credit	3	57.75
Debit	3	126.64
KarmaPoints	12	11581.88
Saving	2	9066.64

3.h. Retrieve the name and contact info of the customer who has the highest karma point balance.

```
SELECT user_name AS buyer_name, user_email AS contact_info,
acct_type AS account_type, MAX(acct_balance) AS balance
FROM(
  SELECT `buyer`.buyer_id, `user`.user_name, `user`.user_id,
`user`.user_email, `payment_method`.acct_type,
`payment_method`.acct_balance
  FROM `buyer`, `user`, `payment_method`
  WHERE `buyer`.buyer_id = `user`.user_id AND
`payment_method`.pay_user_id = `user`.user_id AND
`payment_method`.acct_balance >=0
 ) AS sub;
```

buyer_name	contact_info	account_type	balance
Leo Messi	messi123@gmail.com	KarmaPoints	100

3.i. Create a list of top 10 rated IP items and the stores selling those.

```
SELECT `product`.pro_id AS product_id, `product`.pro_name AS
product_name, `store`.store_name,
AVG(`reviews`.review_item_score) AS average_score
FROM `store`, `product`, `order_item`, `reviews`
WHERE `store`.store_id = `product`.pro_store_id AND
`order_item`.oitem_pro_id = `product`.pro_id AND
`reviews`.review_oitem_id = `order_item`.oitem_id
GROUP BY `product`.pro_id
ORDER BY average_score DESC
LIMIT 10;
```

product_id	product_name	store_name	average_score
6600016	Imax Gold Card	Amex Card	99
660006	David Copperfield	Apple PDF	98
6600017	Jave Code for the best calculator	The Best Code	95
6600014	Calculas I	Study Online	94
660002	Xon Picture 1	Clion Digital	94
660007	Little Prince	Apple PDF	89
6600013	How to cook	Aqima	87
6600015	Photo America	Wolf PDF	83
660008	Fake Digital Book 1	Banana PDF	80
660001	PDF Editor V3.0	Wolf PDF	78

ExtraQueries.sql

4.a Find all products with their store name.

```
SELECT  pro_id AS product_id, pro_name AS product_name,
store_name, store_id
FROM store
LEFT OUTER JOIN product ON pro_store_id = store_id
ORDER BY pro_id;
```

product_id	product_name	store_name	store_id
660001	PDF Editor V3.0	Wolf PDF	800030
660002	Xon Picture 1	Clion Digital	800010
660003	Xon Picture 2	Clion Digital	800010
660004	Xon Book 1	Clion Books	800011
660005	Xon Book 2	Clion Books	800011
660006	David Copperfield	Apple PDF	800051
660007	Little Prince	Apple PDF	800051
660008	Fake Digital Book 1	Banana PDF	800052
660009	Fake Digital Book 2	Pear PDF	800053
6600010	Fake Digital Book 3	Orange PDF	800054
6600011	Star War II	Video games Store	800040
6600012	Star War III	Digital Gamestop	800070
6600013	How to cook	Aqima	800050
6600014	Calculas I	Study Online	800000
6600015	Photo America	Wolf PDF	800030
6600016	Imax Gold Card	Amex Card	800020

4.b List the average price of IPs in all the stores.

```
SELECT store_id, store_name, ROUND(AVG(pro_price),2) AS
average_price
FROM STORE, PRODUCT
WHERE STORE.store_id = PRODUCT.pro_store_id
GROUP BY store_id;
```

product_id	store_name	average_price
800000	Study Online	38.88
800010	Clion Digital	8.38
800011	Clion Books	28.88
800020	Amex Card	99.99
800030	Wolf PDF	19.44
800040	Video games Store	108.88
800050	Aqima	88.88
800051	Apple PDF	14.94
800052	Banana PDF	9.99
800053	Pear PDF	19.99
800054	Orange PDF	38.88
800060	The Best Code	8888.88
800070	Digital Gamestop	138.88
800080	Online Bookstore	18.88
800090	Slience Online Bookstore	28.88
800091	Crazy Online Bookstore	63.88

4.c Count the number of customer service agents.

```
SELECT COUNT(*) AS number_of_agents
FROM CUSTOMER_SERVICE;
```

number_of_agents
5

AdvancedQueries.sql

5.a Provide a list of buyer names, along with the total dollar amount each buyer has spent in the last year.

```
SELECT user_id, user_name, ROUND(SUM(order_price),2)
FROM user, buyer, orders
WHERE user_id = buyer_id AND buyer_id = order_buyer_id
AND order_date > 2020-01-01
GROUP BY user_id;
```

user_id	user_name	SUM(order_price)
10001	Leo Messi	2086.63
10002	Cristiano Ronaldo	9205.519999999999
10004	Joe Biden	136.64
10005	Jack Brim	9127.75
10006	Sage Lee	38.87
10008	Reyna Davis	237.5

5.b Provide a list of buyer names and email addresses for buyers who have spent more than the average buyer.

```
SELECT user_name, user_email, sumPrice
FROM(SELECT user_name, user_email, sumPrice, AVG(sumPrice) AS
average
FROM ( SELECT user_name, user_email, SUM(order_price) AS
sumPrice
  FROM user, buyer, orders
  WHERE user_id = buyer_id AND buyer_id = order_buyer_id
     GROUP BY user_id)
)
WHERE average < sumPrice;
```

user_name	user_email	sumPrice
Cristiano Ronaldo	ronaldo123@gmail.com	9205.519999999999
Jack Brim	brim123@gmail.com	208329.1000000006

5.c Provide a list of the IP Item names and associated total copies sold to all buyers, sorted from the IP Item that has sold the most individual copies to the IP Item that has sold the least.

```
SELECT pro_id, pro_name, COUNT(oitem_id) as item_count
FROM product
LEFT OUTER JOIN order_item
ON pro_id = oitem_pro_id
GROUP BY pro_id
ORDER BY item_count DESC;
```

pro_id	pro_name	item_count
660008	Fake Digital Book 1	3
6600012	Star War III	3
6600014	Calculas I	2
6600017	Jave Code for the best calculator	2
660001	PDF Editor V3.0	1
660002	Xon Picture 1	1
660006	David Copperfield	1
660007	Little Prince	1
6600013	How to cook	1
6600015	Photo America	1
6600016	Imax Gold Card	1
6600020	Fake Digital Book 6	1
6600025	Fake Digital Book 11	1
6600033	Xon Picture 10	1
660003	Xon Picture 2	0
660004	Xon Book 1	0
660005	Xon Book 2	0
660009	Fake Digital Book 2	0
6600010	Fake Digital Book 3	0

5.d Provide a list of the IP Item names and associated dollar totals for copies sold to all buyers, sorted from the IP Item that has sold the highest dollar amount to the IP Item that has sold the smallest.

```
SELECT pro_id, pro_name, SUM(oitem_price) as total_price
FROM product
LEFT OUTER JOIN order_item
ON pro_id = oitem_pro_id
GROUP BY pro_id
ORDER BY total_price DESC;
```

pro_id	pro_name	total_price
6600017	Jave Code for the best calculator	17777.76
6600012	Star War III	416.64
6600016	Imax Gold Card	99.99
6600013	How to cook	88.88
6600025	Fake Digital Book 11	88.88
6600014	Calculas I	77.76
6600020	Fake Digital Book 6	38.88
660008	Fake Digital Book 1	29.97
660006	David Copperfield	28.88
6600015	Photo America	28.88
660007	Little Prince	19.88
660001	PDF Editor V3.0	10
660002	Xon Picture 1	8.88
6600033	Xon Picture 10	8.88

5.e Find the seller who sold the most items.

```
SELECT user_id, user_name, item_count
FROM(
      SELECT user_id, user_name, COUNT(oitem_id) AS
item_count
      FROM user, seller, product, order_item
      WHERE user_id = seller_id AND seller_id =
pro_seller_id AND pro_id = oitem_pro_id
      GROUP BY user_id)
ORDER BY item_count DESC
LIMIT 1;
```

user_id	user_name	item_count
90005	Iric Sunny	6

5.f Find the most profitable seller

```
SELECT user_id, user_name, total_price
FROM(
      SELECT user_id, user_name, SUM(oitem_price) AS
total_price
      FROM user, seller, product, order_item
      WHERE user_id = seller_id AND seller_id =
pro_seller_id AND pro_id = oitem_pro_id
      GROUP BY user_id)
ORDER BY total_price DESC
LIMIT 1;
```

user_id	user_name	total_price
90006	Mark Smith	17777.76

5.g Provide a list of buyer names for buyers who purchased anything listed by the most profitable seller.

```
SELECT user_id, user_name
FROM user,buyer, orders, order_item, product, seller
WHERE user_id = buyer_id AND order_buyer_id = buyer_id
AND order_id = oitem_order_id AND oitem_pro_id = pro_id
AND pro_seller_id = seller_id AND seller_id = (SELECT
user_id
FROM(
      SELECT user_id, user_name, SUM(oitem_price) AS
total_price
      FROM user, seller, product, order_item
      WHERE user_id = seller_id AND seller_id =
pro_seller_id AND pro_id = oitem_pro_id
      GROUP BY user_id)
ORDER BY total_price DESC
LIMIT 1);
```

user_id	user_name
10002	Cristiano Ronaldo
10005	Jack Brim

5.h Provide the list of sellers who listed the IP Items purchased by the buyers who have spent more than the average buyer.

```
SELECT user_id, user_name
FROM user, seller, product, order_item, orders, buyer
WHERE user_id = seller_id AND seller_id = pro_seller_id AND
pro_id = oitem_pro_id AND order_id = oitem_order_id AND
buyer_id = order_buyer_id AND buyer_id =   (SELECT user_id
FROM ( SELECT user_name, user_email, sumPrice, user_id
FROM(SELECT user_name, user_email, sumPrice, AVG(sumPrice) AS
average, user_id
FROM ( SELECT user_name, user_email, SUM(order_price) AS
sumPrice, user_id
  FROM user, buyer, orders
  WHERE user_id = buyer_id AND buyer_id = order_buyer_id
     GROUP BY user_id)
)
WHERE average < sumPrice)
);
```

user_id	user_name
90006	Mark Smith

5.i Provide sales statistics (number of items sold, highest price, lowest price, and average price) for each type of IP item offered by a particular store.

```
SELECT product.pro_category, item_count,
ROUND(MAX(pro_price),2), MIN(pro_price), AVG(pro_price)
FROM store, product,(SELECT pro_category, COUNT(oitem_id) AS
item_count
FROM store, product, order_item
WHERE store_id = pro_store_id AND oitem_pro_id = pro_id AND
store_id = 800010
GROUP BY product.pro_category) AS sub
WHERE store_id = pro_store_id AND store_id = 800010 AND
sub.pro_category = product.pro_category
GROUP BY product.pro_category;
```

pro_category	item_count	MAX(pro_price)	MIN(pro_price)	AVG(pro_price)
image	2	28.88	0.88	9.102222222222222

S2.C INSERT and DELETE SQL code samples.

INSERT samples

1. INSERT INTO USER

```
INSERT INTO `user` (user_id, user_name, user_DOB,
user_gender, user_email)
VALUES  ('90001','Lebron
James','1984-12-30','M','james123@gmail.com'),
          ('90002','Kyrie
Irving','1992-03-23','F','irving123@gmail.com');

INSERT INTO `user` (user_id, user_name, user_DOB,
user_gender, user_email)
VALUES  ('90003','Serena
Williams','1981-09-26','F','williams123@gmail.com');

INSERT INTO `user` (user_id, user_name, user_DOB,
user_gender, user_email)
VALUES  ('90004','Taylor
Yang','2000-12-30','M','taylor@outlook.com');
```

user_id	user_name	user_DOB	user_gender	user_email
90001	Lebron James	1984-12-30	M	james123@gmail.com
90002	Kyrie Irving	1992-03-23	F	irving123@gmail.com
90003	Serena Williams	1981-09-26	F	williams123@gmail.com
90004	Taylor Yang	2000-12-30	M	taylor@outlook.com

2. INSERT INTO PRODUCTS

```
INSERT INTO `product` (pro_id, pro_name, pro_category,
pro_stock, pro_price, pro_store_id, pro_seller_id, pro_qi_id)
VALUES  ('660001','PDF Editor V3.0','software','500','10',
'800030', '90003', '7601'),
          ('660002','Xon Picture 1','image','999','8.88',
'800010', '90001', '7601'),
        ('660003','Xon Picture 2','image','999','28.88',
'800010', '90001', '7602'),
        ('660004','Xon Book 1','book','500','18.88',
'800011', '90001', '7603'),
        ('660005','Xon Book 2','book','499','38.88',
'800011', '90001', '7604'),
        ('660006','David Copperfield','book','999','9.99',
'800051', '90005', '7604');
```

pro_id	pro_name	pro_category	pro_stock	pro_price	pro_store_id	pro_seller_id	pro_qi_id
660001	PDF Editor V3.0	software	500	10	800030	90003	7601
660002	Xon Picture 1	image	999	8.88	800010	90001	7601
660003	Xon Picture 2	image	999	28.88	800010	90001	7602
660004	Xon Book 1	book	500	18.88	800011	90001	7603
660005	Xon Book 2	book	499	38.88	800011	90001	7604
660006	David Copperfield	book	999	9.99	800051	90005	7604

3. INSERT INTO PRODUCT_IMAGES

```
INSERT INTO `pro_images` (pro_image_url, images_id,
images_pro_id)
VALUES  ('https://link1_1.com', '1', '660001'),
          ('https://link1_2.com', '2', '660001'),
          ('https://link1_3.com', '3', '660001'),
          ('https://link2.com', '1', '660002'),
        ('https://link5.com', '1', '660005');
```

pro_image_url	images_id	images_pro_id
https://link1_1.com	1	660001
https://link1_2.com	2	660001
https://link1_3.com	3	660001
https://link2.com	1	660002
https://link5.com	1	660005

DELETE samples:

1. DELETE FROM USER

```
DELETE FROM `user` WHERE user_id = 90001;
DELETE FROM `user` WHERE user_id = 90002;
DELETE FROM `user` WHERE user_id = 90003;
```

user_id	user_name	user_DOB	user_gender	user_email
90004	Taylor Yang	2000-12-30	M	taylor@outlook.com

2. DELETE FROM PRODUCTS

```
DELETE FROM `product` WHERE pro_id = 660005;
```

pro_id	pro_name	pro_category	pro_stock	pro_price	pro_store_id	pro_seller_id	pro_qi_id
660001	PDF Editor V3.0	software	500	10	800030	90003	7601
660002	Xon Picture 1	image	999	8.88	800010	90001	7601
660003	Xon Picture 2	image	999	28.88	800010	90001	7602
660004	Xon Book 1	book	500	18.88	800011	90001	7603
660006	David Copperfield	book	999	9.99	800051	90005	7604

Since product_images is a weak entity of product, if parent is deleted(id=660005), then it's children will also be deleted(pro_image_660005)automatically.
(This delete cascade may not be able to show at sqlite.online.)

https://link1_1.com	1	660001
https://link2.com	1	660002
https://link3.com	1	660003
https://link4.com	1	660004
https://link6.com	1	660006

product(id=660006) cannot be deleted since there is an order(s) related to this product. To prevent further issues (e.x. Quality issue - sellers can delete it without further inspection), involved products could only be deleted when no order/refund request related to them.

	Time	Action
1	21:01:18	SELECT * FROM `Bits&Bots_TEAM2`.order_item LIMIT 0, 1000
2	21:01:38	DELETE FROM `product` WHERE pro_id = 660006
3	21:01:49	DELETE FROM `product` WHERE pro_id = 660005
4	21:01:58	SELECT * FROM `Bits&Bots_TEAM2`.pro_images LIMIT 0, 1000

3. DELETE FROM PRODUCT_IMAGES

```
DELETE FROM `pro_images` WHERE iamges_pro_id = 660001;
```

pro_image_url	images_id	images_pro_id
https://link2.com	1	660002

S2.D Two indexes properly explained,including SQL code.

1. From the queries, ORDERS is constantly joined with USER to find orders for specific users. Therefore, creating an index on userID would greatly help speed up the join condition between USER and ORDERS. Hash-base index is better in this case because user_id is mainly used for equality tests instead of range tests.

```
CREATE UNIQUE INDEX idx_userID
ON `user` (user_id);
```

2. Often we need to search for a product under a certain price. Then having an index on price would be very helpful. Range tests usually perform on product price so a tree-base index is preferred in this case.

```
CREATE UNIQUE INDEX idx_pro_price
ON `product` (pro_price);
```

S2.E Two Views Explained,including SQL code data resulting from the execution.

1. Create a view to show all sellers with their store name and their total value of products.

```
CREATE VIEW SELLER_INFO (Seller_name, Store_name,
Total_value)
AS SELECT user_name, store_name,
ROUND(SUM(pro_price*pro_stock),2)
      FROM USER, SELLER, STORE, PRODUCT
      WHERE user_id = seller_id AND seller_id =
store_seller_id AND store_id = pro_store_id
      GROUP BY user_name;
```

Seller_name	Store_name	Total_value
Alice Brown	Slience Online Bookstore	274883.84
Iric Sunny	Apple PDF	109553.39
James Harden	Digital Gamestop	13749.12
Jennis Jones	Study Online	7737.12
Kyrie Irving	Amex Card	9899.01
Lebron James	Clion Digital	112557.32
Mark Smith	The Best Code	4435551.12
Serena Williams	Wolf PDF	10747.12
Taylor Yang	Video games Store	21667.12
Yaqian Wu	Online Bookstore	3757.12

2. Create a view to show all buyers with their order counts and total money spent.

```
CREATE VIEW BUYER_INFO (Buyer_name, Counts, Spending)
AS SELECT user_name, COUNT(*), ROUND(SUM(order_price),2)
      FROM USER, BUYER, ORDERS
      WHERE user_id = buyer_id AND buyer_id = order_buyer_id
      GROUP BY user_name;
```

Buyer_name	Counts	Spending
Cristiano Ronaldo	3	9205.52
Jack Brim	3	9127.75
Joe Biden	3	136.64
Leo Messi	5	2086.63
Reyna Davis	4	237.5
Sage Lee	2	38.87

S2F. Two transactions explained, including SQL code.

1. One transaction would be a buyer placing an order. The transaction includes inserting an order into ORDERS, updating the account balance, and updating stock quantity of the product if it is countable.

```
BEGIN TRANSACTION;

INSERT OR ROLLBACK INTO ORDERS
VALUES('4400042','2021-07-09','biden123@gmail.com','88.8
8','10004', '4', '410004');

UPDATE OR ROLLBACK PAYMENT
SET AcctBalance = AcctBalance - 88.88
WHERE userID = '10001';

UPDATE OR ROLLBACK PRODUCT
SET stockQuantity  = stockQuantity - 1
WHERE proID = '600025';

COMMIT;
```

2. New users need to create an account. This transaction includes inserting a user account in USER and inserting a new payment account into PAYMENT.

```
BEGIN TRANSACTION;

INSERT OR ROLLBACK INTO USER
VALUES ('100011', 'John Willims', '2000-01-01', 'M',
'john123@gmail.com');

INSERT OR ROLLBACK INTO PAYMENT
VALUES ('2','400966623',NULL,'2025-09-15',NULL,'Credit',
'100011');
```

```
COMMIT;
```

Section 3 -- Team reports and Graded Checkpoint Documents

a. Detailed description of all team member contributions

The final project is roughly divided into a few portions for each team member. Nicholas has done most of the database creation and project formatting work. Nick has also helped with queries checking and group coordination. Peitong mostly did the data insertion for the database and project summary. Taiyi did the work on database queries, indexes, views and transactions. Minhye worked on the function table creation and proofread the group work. For checkpoints, work was usually done on a rolling basis. We started right after the completion of the topic and each member would try to finish as much work before the last day. And on the last day, we would get together to finish the remaining work.

b. Reflection on the project completion process

We started working on the checkpoint as soon as the professor had covered the material in class. Due to the time zone difference, we were unable to get together to work together for the most part. The work is roughly divided and each member would finish as much as they can. On the due day we would get together to check the finished work and work the remaining part. For the final project, it is mainly gathering information from the previous checkpoint with some updates. We finished all the work in the final week.

c. Description of feedback received, and revisions completed throughout the process

The feedback among the group once received. Then on the next group gathering before the next checkpoint, we would go through the feedback and try to fix the errors. Sometimes we went to the professor's office hours for clarification. For the final project we make sure we resolve every feedback we have received.

d. Marked Project Checkpoints and Worksheets

See `CP_WORK_FEEDBACK` folder.

Part II --The SQL Database (README)

ROOT FOLDER
FINAL_PROJECT_TEAM_2.PDF
FINAL_PROJECT_TEAM_2.DOCX

FINAL_ERD_TEAM_2.PNG
FINAL_SCHEMA_TEAM_2.PNG

SQL_TEAM_2 FOLDER
CreateQueries.txt --- USE FIRSTLY
InsertQueries.txt --- USE SECONDLY
SimpleQueries.txt
ExtraQueries.txt
AdvancedQueries.txt

CP_WORK_FEEDBACK FOLDER --- CP01-CP04

EXTERNAL LINK
https://drive.google.com/file/d/1aqOwf0lWnXxBnASTyd1IJCcxB6T7O0MS/view?usp=sharing (EERD)

https://drive.google.com/file/d/1nV6ByEymFAYSMubiAAUTvHnDHgevy4fV/view?usp=sharing (SCHEMA)

SEE .ZIP FILE IN CARMEM ATTACHMENT.