**CSE 3241 FINAL PROJECT REPORT**

**Table of Contents**

**SECTION 1 - Database Description (Logical DB design)**

a. Introduction and project summary Pg. 3

b. (E)ERD – Model properly documented and explained. Pg. 5

c. Relational schema properly documented and explained. Pg. 7

d. Relational algebra statements for all SELECT queries from CP02 Pg. 11

e. Database fully normalized, with correct justifications Pg. 13

**SECTION 2 - User Manual (DB Implementation)**

a. Table description including table functions, keys, constraints, and data types. Pg. 20

b. A catalog of supplied SQL Queries with explanations and sample output for each. Pg. 28

c. INSERT and DELETE SQL code samples. Pg. 35

d. Two indexes properly explained, including SQL code. Pg. 36

e. Two views explained, including SQL code and data resulting from the execution. Pg. 37

f. Two transactions explained, including SQL code. Pg. 38

**SECTION 3 - Team Reports and Graded Checkpoint Documents**

a. Detailed description of all team member contributions. Pg. 39

b. Reflection on project completion process. Pg. 40

c. Description of feedback received, and revisions completed throughout the process. Pg. 41

d. Marked Project Checkpoints and Worksheets. Pg.

**1a Introduction and Project Summary**

In this project, we created a relational database for DB 4Ever’s new website Bits & Bots. We designed the database to fulfill the basic requirements provided to us, such as supporting virtual inventory, buyer/seller accounts, sales, and feedback operations. We also added some extra functionality not described in the basic requirements, most notably in creating entities for buyer’s to have a wishlist, and to create categories for sellers to place their virtual stores into.

We address the needs of our client in the following ways. First, we will discuss the needs that a buyer would have on this website. We have an entity that consists of each IP-Item listed in the store. This is accompanied with attributes such as the name, price, and a description of the item. Thus, any buyer can search through a large catalog of files for what they need. When buyers choose the items they would like to purchase, they can have their order consist of multiple items from multiple different stores and sellers. Each order will generate a record, which will be linked to multiple smaller records that consist of the order id of the order and the item id of the item. This allows for seamless and convenient ordering and record keeping. Buyers can also use multiple payment methods. A buyer can choose a certain payment method and the amount to pay with that payment method. This can be done multiple times per order, thus the functionality exists to pay for a single order with multiple payment methods. And, a buyer can specify which email address to send an order to, by setting the order email on their order.

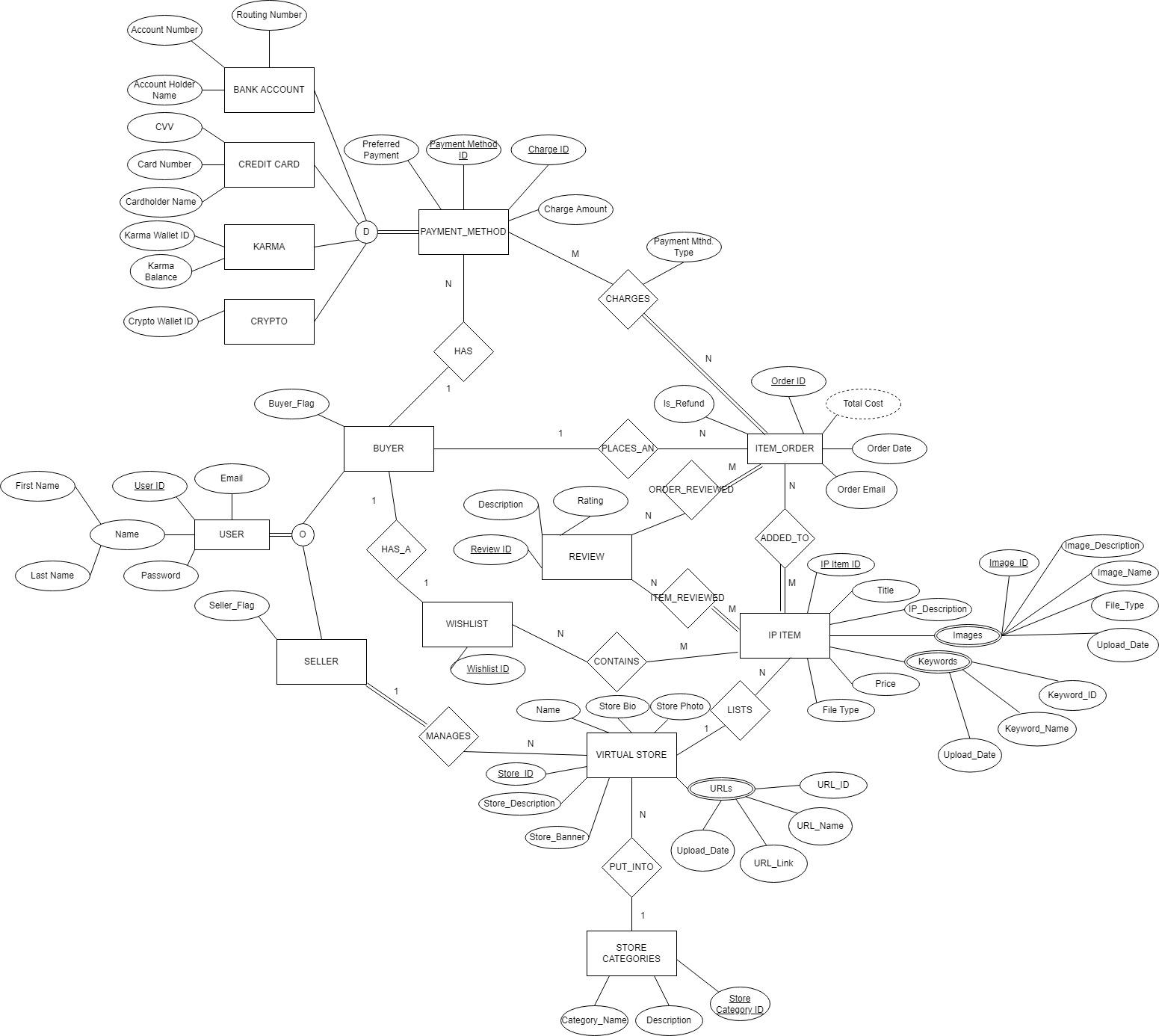
Next, we will discuss the needs for a seller on the website. A seller is able to create a listing for their IP\_Item. As previously stated, the IP\_Item entity will contain all products listed on the site. The seller can set attributes such as a title, description, price, can display screenshots/images, can list keywords, and can specify the file type. Sellers can also create virtual stores, which can be used to organize the items that they list on the website. For their virtual stores, sellers have the ability to set attributes such as its name, description, banner, a bio, a photo, and URLS for their personal links. They can create multiple websites on the site.

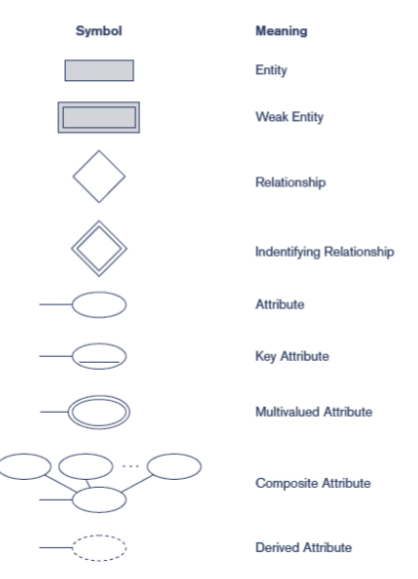
Finally, we will discuss what we have implemented to benefit both buyers and sellers. A review system is in place that will allow buyers to review products that they have purchased. This is done by having our Review entity have relationships with IP\_Item, to specify the item being reviewed, and Order, to make sure that the buyer does have an order that lists the item. This benefits both buyers and sellers, by creating a fair review system. There is also a financial management and reporting portal, which will allow the seller to track sales, gains, and losses, and the buyer to track their purchases and refunds, among other things. There is also a system in place so that sellers can issue, and buyers can receive, refunds. This is indicated by the Is\_Refund flag on each order. If this flag is activated, the transaction will be a refund.

The last thing we will discuss is the benefits of the extra features we have added in addition to the requirements. The wishlist is a feature for buyers, but will benefit sellers and the site itself greatly, by allowing buyers to keep track of items they are interested in. With this functionality, buyers can easily access the listings for products they like, and sellers can see which items are wishlisted the most, allowing them to see what is most popular with our community of buyers. The second additional feature is our store categories. This is another way for buyers to easily locate products based on what they are interested in, and also allow sellers to make their shops easier to find, either by selecting popular categories, or selecting underutilized categories that they believe there is demand for. We believe that our additional features, as well as the design of our database as a whole, provides a solid foundation for an e-commerce website, creating a lot of potential to drive sales and create a good user experience.

**1b EERD Model**

Below is our EERD and our key.





For the EERD, there were a lot of considerations taken into account. We first decided to create the USER entity and create an overlapping subclass, resulting in a BUYER and SELLER entity. Whenever there was a new user, they would be flagged via the buyer or seller flag. This decision then made it simpler so that only buyers could place orders, leave reviews, and add their payment methods. Sellers could then manage their stores and the items they have available. For when a user wants to purchase an item, they must have their payment methods linked to their account. For the PAYMENT\_METHOD entity, we decided to create a disjoint subclass, possessing each specific payment method type. The subclass contained BANK\_ACCOUNT, CREDIT\_CARD, KARMA, and CRYPTO. We decided to do a subclass for the payment methods, due to their being four different types and these types differed in how they were handled. Bank accounts and credit cards required pretty similar properties, but needing to satisfy crypto and karma purchases were our deciding factors into going to an overlapping subclass.

**1c Relational Schema**

Our relational schema is as follows.

Preferred payment and charge amount?

**USER** (User\_ID, First\_Name, Last\_Name, Password, Buyer\_Flag, Seller\_Flag, Email)

User\_ID is the primary key for user and it does not possess any foreign keys.

**WISHLIST** (Wishlist\_ID**,** User\_ID (FK))

User ID is a foreign key originating from **USER**.

Relation has the foreign key User\_ID due to 1:1 relationship with BUYER.

**ITEMS\_IN\_WISHLIST** (Wishlist\_ID\_(FK), IP\_Item\_ID (FK))

Wishlist ID and IP Item ID are foreign keys originating from **WISHLIST** and **IP ITEM**, respectively.

Relation created because of N:M relationship with WISHLIST and IP ITEM. Therefore it contains both Wishlist\_ID and IP\_Item\_ID as foreign keys.

**REVIEW** (Review\_ID, Rating, Description)

Relation has Review ID as primary key and contains no foreign keys.

**BANK ACCOUNT** (Payment Method ID (FK), Account Holder Name, Account Number, Routing Number, Charge ID (FK), User ID (FK))

User ID is a foreign key originating from **USER** and Payment Method ID and Charge ID are foriegn keys originating from **PAYMENT\_METHODS**.

Relation has the key Payment Method ID as a primary key and foreign key due to it being a disjoint subclass of PAYMENT\_METHOD. Has the key Charge ID as a foreign key due to it being a primary key in the entity PAYMENT\_METHOD. Has the key User ID as a foreign key due to the 1:N relationship between BUYER and PAYMENT METHODS.

**CREDIT CARD** (Payment Method ID (FK), CVV, Card Number, Cardholder Name, Charge ID (FK), User ID (FK))

User ID is a foreign key originating from **USER** and Payment Method ID and Charge ID are foriegn keys originating from **PAYMENT\_METHODS**.

Relation has the key Payment Method ID as a primary key and foreign key due to it being a disjoint subclass of PAYMENT\_METHOD. Has the key Charge ID as a foreign key due to it being a primary key in the entity PAYMENT\_METHOD. Has the key User ID as a foreign key due to the 1:N relationship between BUYER and PAYMENT METHODS.

**KARMA** (Payment Method ID (FK), Karma Wallet ID, Karma Balance, Charge ID (FK), User ID (FK))

User ID is a foreign key originating from **USER** and Payment Method ID and Charge ID are foriegn keys originating from **PAYMENT\_METHODS**.

Relation has the key Payment Method ID as a primary key and foreign key due to it being a disjoint subclass of PAYMENT\_METHOD. Has the key Charge ID as a foreign key due to it being a primary key in the entity PAYMENT\_METHOD. Has the key User ID as a foreign key due to the 1:N relationship between BUYER and PAYMENT METHODS.

**CRYPTO** (Payment Method ID (FK), Crypto Wallet ID, Charge ID (FK), User ID (FK))

User ID is a foreign key originating from **USER** and Payment Method ID and Charge ID are foriegn keys originating from **PAYMENT\_METHODS**.

Relation has the key Payment Method ID as a primary key and foreign key due to it being a disjoint subclass of PAYMENT\_METHOD. Has the key Charge ID as a foreign key due to it being a primary key in the entity PAYMENT\_METHOD. Has the key User ID as a foreign key due to the 1:N relationship between BUYER and PAYMENT METHODS.

**ITEM\_ORDER** (Order ID, Order Date, Is Refund, Order Email, User ID (FK))

User ID is a foreign key originating from **USER**.

Relation has the key User ID as a foreign key due to 1:N relationship between BUYER and ITEM\_ORDER.

**CHARGES** (Order ID (FK), Payment Method ID (FK), Payment Mthd. Type)

Order ID and Payment Method Type are both foreign keys originating from **ITEM\_ORDER** and **PAYMENT\_METHODS**, respectively.

Relation created from N:M relationship with ITEM\_ORDER and PAYMENT\_METHODS. Therefore it contains Order ID and Payment Method ID as foreign keys.

**ITEM\_ADDED\_TO\_ORDER** (Order ID (FK), IP Item ID (FK))

Order ID and IP Item ID are both foreign keys originating from **ORDER** and **IP ITEM**, respectively.

Relation created from N:M relationship with IP\_ITEM and ORDER. Therefore it contains Order ID and IP Item ID as foreign keys.

**IP ITEM** (IP\_Item\_ID, Store\_ID (FK), Title, IP\_Description, Price, File\_Type)

Store ID is a foreign key originating from **VIRTUAL STORE**.

Relation has the key Store\_ID as a foreign key due to 1:N relationship between IP ITEM and VIRTUAL STORE.

**IP ITEM IMAGES** (Image\_ID, IP\_Item\_ID (FK), Image\_Description, Image\_Name, File\_Type, Upload\_Date)

IP Item ID is a foreign key originating from **IP ITEM**.

Relation has the key IP Item ID as a foreign key due to images being a multivalued attribute of IP ITEM.

**IP ITEM KEYWORDS** (Keyword\_ID, IP\_Item\_ID (FK), Keyword\_Name, Upload\_Date)

IP Item ID is a foriegn key originating from **IP ITEM**.

Relation has the key IP Item ID as a foreign key due to images being a multivalued attribute of IP ITEM.

**VIRTUAL STORE** (Store ID, User\_ID (FK), Store\_Description, Store\_Banner, Name, Store\_Photo, Store\_Bio, Store\_Category ID (FK))

User ID is a foreign key originating from **USER**.

Store\_Category is a foreign key originating from Category Name in **STORE CATEGORIES.**

Relation has the key User ID as a foreign key due to 1:N relationship between SELLER and VIRTUAL STORE. Has the key Store Category ID as a foreign key due to 1:N relationship between STORE CATEGORIES and VIRTUAL STORE.

**VIRTUAL STORE URLS** (URL\_ID, Store\_ID (FK), URL\_Name, URL\_Link, Upload\_Date)

Store ID is a foreign key originating from **VIRTUAL STORE**.

Relation has the key Store ID as a foreign key due to URLs being a multivalued attribute of VIRTUAL STORE.

**IP ITEM REVIEWED** (IP Item ID, Review ID)

IP Item ID is a foreign key originating from **IP ITEM**

Review ID is a foregin key originating from **REVIEW**

Relation created from N to M relationship with IP\_ITEM and REVIEW. Therefore it contains Review ID and IP Item ID as foreign keys.

**ORDER REVIEWED** (Order ID, Review ID)

Order ID is a foreign key originating from **ORDER**

Review ID is a foreign key originating from **REVIEW**

Relation created from N:M relationship with REVIEW and ORDER. Therefore it contains Order ID and Review ID as foreign keys.

**STORE CATEGORIES** (Store\_Category ID, Category\_Name, Description)

Store ID is a foreign key originating from **VIRTUAL STORE**.

Relation has Store Category ID as a primary key and no foreign keys.

**1d Relational Algebra Statements**

The following are relational algebra statements with explanations of what each statement does:

**The statement below creates a list of all the IP Items and the name of each store that is selling each specific item:**

IP\_ITEMS\_AND\_STORES ← VIRTUAL\_STORE \* IP\_ITEM

*π*(Title, Name)(IP\_ITEMS\_AND\_STORES)

**The statement below searches the database for IP Items that cost less than $10 and creates a list showing each item and its price:**

*πtitle (σ*(Price < 10)(IP\_Item))

**The statement below searches the database for a specific user, based upon User\_ID. The statement then finds the name of each IP Item they have purchased along with the date of purchase and creates a list showing both the title and date:**

SOME\_USER ← *σ*(User ID = 0x64bb757d)(Order)

SELECTED\_USER ← SOME\_USER \* ITEM\_ADDED\_TO\_ORDER

SELECTED\_USER\_ORDER\_IP ← SELECTED\_USER \* IP\_ITEM

*π*(Title, Order Date)SELECTED\_USER\_ORDER\_IP

**The statement below searches the database for a specific Store, based upon Store\_ID. The statement then finds the first and last name of each buyer who purchased from the store, and the title of the IP Item’s each buyer has purchased, then creates a list of these attributes:**

SOME\_STORE ← *σ*(Store ID = 19384768)(IP\_ITEM)

SELECTED\_STORE ← SOME\_STORE \* ITEM\_ADDED\_TO\_ORDER

SELECTED\_STORE\_ITEM\_IP ← SELECTED\_STORE \* ORDER

FINAL\_IP\_ITEM\_BUYER\_LIST ← SELECTED\_STORE\_ITEM\_UP \* USER

*π*(First Name, Last Name, Title)FINAL\_IP\_ITEM\_BUYER\_LIST

**The statement below counts each IP Item purchased by a User, then finds the Max of those counts. After the Max is found, the User who has purchased the most Items is then listed along with the number of purchases they had:**

SOME\_ORDER ← ITEM\_ADDED\_TO\_ORDER \* ORDER

SOME\_USER\_ID(User\_ID, Items\_Purchased) ←User IDFCOUNT IP\_ITEM(SOME\_ORDER)

MOST\_ITEM\_BOUGHT(User ID, Max\_Items\_Purchased) ← User IDFMAX Items\_Purchased (SOME\_USER\_ID)

**The statement below counts the amount of IP Items each Store has, then if a Store has 5 or less items, the Store name is listed alongside the number of items it has:**

STORES\_BY\_SOLD\_ITEMS(Store ID, Item Count) ← Store IDFCOUNT IP Item ID(IP\_ITEM)

*σ*Item Count <= 5(STORES\_BY\_SOLD\_ITEMS)

**The statement below calculates a count of each time an IP Item is bought, then sums the total sales from each item. The item with the highest amount of dollar sales is found and lists the units sold, total dollar sales, and the seller of the item:**

IP\_ITEM\_SALES <- IP\_ITEM \* ITEM\_ADDED\_TO\_ORDER

ITEM\_COUNT(IP\_Item\_ID, Quantity, Sales) ← IP Item IDFCOUNT IP Item ID(IP\_ITEM\_SALES)

ITEM\_SALES(IP\_Item\_ID, Sales) <- IP Item IDFSUM Price(IP\_ITEM\_SALES)

MAX\_ITEM\_COUNT(IP\_Item\_ID, Quantity) <- Item IDFMAX Quantity(ITEM\_COUNT)

MAX\_ITEM(IP\_Item\_ID, Quantity, Sales) <- MAX\_ITEM\_COUNT \* ITEM\_SALES

MAX\_ITEM\_STORE(IP\_Item\_ID, Quantity, Sales, Name) <- MAX\_ITEM \* VIRTUAL\_STORE

**The statement below finds all the payment types that are accepted and the number of times used along with the total amount charged for that payment type:**

CHARGES\_BY\_PAYMENT\_TYPE(Payment Types, Num Payment Types) ← Payment TypesFCOUNT Charge ID(Charges)

CHARGE\_SUMS\_BY\_PAYMENT\_TYPE(Payment Types, Total Charge) ← Payment TypesFSUM Charge Amount(Charges)

ACCEPTED\_PAYMENT\_TYPES ← *π*Payment Types, Num Payment Types, Total Charge(CHARGE\_SUMS\_BY\_PAYMENT\_TYPE \* CHARGES\_BY\_PAYMENT\_TYPE)

**The statement below finds the User with the highest Karma point balance and lists their name and balance:**

USER\_KARMA\_BAL(Max\_Karma\_Balance) ←MAX Karma\_Balance(KARMA)

USER\_INFO\_KARMA(First\_Name, Last\_Name, Email, Max\_Karma\_Balance) <- USER\_KARMA\_BAL \* USER

**The statement below counts the number of stores in each store category and lists them:**

Category NameFCOUNT Store ID(STORE\_CATEGORY)

**The statement below finds an IP Item and how counts how many Wishlists the IP Item belongs to:**

IP Item IDFCOUNT Wishlist ID(ITEMS\_IN\_WISHLIST)

**The statement below finds the average price of each IP Item for each Virtual Store:**

IP Item IDFAVERAGE Price(VIRTUAL\_STORE)

**1e Normalization of Database**

**USER** (User\_ID, First\_Name, Last\_Name, Password, Buyer\_Flag, Seller\_Flag, Email)

User’s functional dependencies are the following:

{User\_ID} -> {First\_Name, Last\_Name, Password, Buyer\_Flag, Seller\_Flag, Email}

{Email} -> {User\_ID, First\_Name, Last\_Name, Password, Buyer\_Flag, Seller\_Flag}

It is in 1NF because it meets all four requirements for 1NF, including that all values are atomic.

It is in 2NF because there are no partial dependencies, as there is only one determinant.

It is in 3NF because there are no transitive dependencies, as there is only one determinant.

It is in BCNF because there are no attributes that are dependent on a non-primary key. Email is also a candidate key, since it is unique for each user.

**ITEMS\_IN\_WISHLIST** (Wishlist\_ID\_(FK), IP\_Item\_ID (FK))

Wishlist ID and IP Item ID are foreign keys originating from **WISHLIST** and **IP ITEM**, respectively.

Items\_In\_Wishlist’s functional dependencies are the following:

{Wishlist\_ID, IP\_Item\_ID}

It is in 1NF because it meets all four requirements for 1NF, including that all values are atomic.

It is in 2NF because there are no partial dependencies.

It is in 3NF because there are no transitive dependencies.

It is in BCNF because there are no attributes that are dependent on a non-primary key.

**WISHLIST** (Wishlist\_ID**,** User\_ID (FK))

User ID is a foreign key originating from **USER**.

Wishlist’s function dependencies are the following:

{Wishlist\_ID} -> {User\_ID}

{User\_ID} -> {Wishlist\_ID}

It is in 1NF because it meets all four requirements for 1NF, including that all values are atomic.

It is in 2NF because there are no partial dependencies, as there is only one determinant.

It is in 3NF because there are no transitive dependencies.

It is in BCNF because there are no attributes that are dependent on a non-primary key.

**REVIEW** (Review\_ID, Rating, Description)

Review’s functional dependencies are the following:

{Review\_ID} -> {Rating, Description}

It is in 1NF because it meets all four requirements, including that all values are atomic.

It is 2NF because there are no partial dependencies, as there is only one determinant.

It is 3NF because there are no transitive dependencies, as there is only one determinant.

It is BCNF because there are no attributes that are dependent on a non-primary key.

**BANK ACCOUNT** (Payment Method ID (FK), Account Holder Name, Account Number, Routing Number, Charge ID (FK), User ID (FK))

User ID is a foreign key originating from **USER** and Payment Method ID and Charge ID are foriegn keys originating from **PAYMENT\_METHODS**.

Bank Account’s functional dependencies are the following:

{Payment\_Method\_ID} -> {Account Holder Name, Account Number, Routing Number, Charge ID, User ID}

It is in 1NF because it meets all four requirements, including that all values are atomic.

It is 2NF because there are no partial dependencies, as there is only one determinant.

It is 3NF because there are no transitive dependencies, as there is only one determinant.

It is BCNF because there are no attributes that are dependent on a non-primary key.

**CREDIT CARD** (Payment Method ID (FK), CVV, Card Number, Cardholder Name, Charge ID (FK), User ID (FK))

User ID is a foreign key originating from **USER** and Payment Method ID and Charge ID are foriegn keys originating from **PAYMENT\_METHODS**.

Credit Card’s functional dependencies are the following:

{Payment\_Method\_ID} -> {CVV, Card Number, Cardholder Name, Charge ID, User ID}

It is in 1NF because it meets all four requirements, including that all values are atomic.

It is 2NF because there are no partial dependencies, as there is only one determinant.

It is 3NF because there are no transitive dependencies, as there is only one determinant.

It is BCNF because there are no attributes that are dependent on a non-primary key. Credit card numbers will be unique, so Card\_Number is a candidate key.

**KARMA** (Payment Method ID (FK), Karma Wallet ID, Karma Balance, Charge ID (FK), User ID (FK))

User ID is a foreign key originating from **USER** and Payment Method ID and Charge ID are foriegn keys originating from **PAYMENT\_METHODS**.

Karma’s functional dependencies are the following:

{Payment Method ID} -> {Karma Wallet ID, Karma Balance, Charge ID, User ID}

It is in 1NF because it meets all four requirements, including that all values are atomic.

It is 2NF because there are no partial dependencies, as there is only one determinant.

It is 3NF because there are no transitive dependencies, as there is only one determinant.

It is BCNF because there are no attributes that are dependent on a non-primary key. Karma\_Wallet\_ID is unique, so Karma\_Wallet\_ID is a candidate key.

**CRYPTO** (Payment Method ID (FK), Crypto Wallet ID, Charge ID (FK), User ID (FK))

User ID is a foreign key originating from **USER** and Payment Method ID and Charge ID are foriegn keys originating from **PAYMENT\_METHODS**.

Crypto’s functional dependencies are the following:

{Payment Method ID} -> {Crypto wallet ID, Charge ID, User ID}

It is in 1NF because it meets all four requirements, including that all values are atomic.

It is 2NF because there are no partial dependencies, as there is only one determinant.

It is 3NF because there are no transitive dependencies, as there is only one determinant.

It is BCNF because there are no attributes that are dependent on a non-primary key. Crypto\_Wallet\_ID is unique, so it is a candidate key.

**ITEM\_ORDER** (Order ID, Order Date, Is Refund, Order Email, User ID (FK))

User ID is a foreign key originating from **USER**.

Item\_Order’s functional dependencies are the following:

{Order\_ID} -> {Order Date, Is Refund, Order Email, User ID}

It is in 1NF because it meets all four requirements, including that all values are atomic.

It is 2NF because there are no partial dependencies, as there is only one determinant.

It is 3NF because there are no transitive dependencies, as there is only one determinant.

It is BCNF because there are no attributes that are dependent on a non-primary key.

**CHARGES** (Order ID (FK), Payment Method ID (FK), Payment Mthd. Type)

Order ID and Payment Method Type are both foreign keys originating from **ITEM\_ORDER** and **PAYMENT\_METHODS**, respectively.

Charges’ functional dependencies are the following:

{Order ID, Payment Method ID} -> {Payment Mthd. Type}

It is in 1NF because it meets all four requirements, including that all values are atomic.

It is 2NF because there are no partial dependencies.

It is 3NF because there are no transitive dependencies.

It is BCNF because there are no attributes that are dependent on a non-primary key.

**ITEM\_ADDED\_TO\_ORDER** (Order ID (FK), IP Item ID (FK))

Order ID and IP Item ID are both foreign keys originating from **ORDER** and **IP ITEM**, respectively.

Item\_Added\_To\_Order’s functional dependencies are the following:

{Order ID, IP Item ID}

It is in 1NF because it meets all four requirements, including that all values are atomic.

It is 2NF because there are no partial dependencies.

It is 3NF because there are no transitive dependencies.

It is BCNF because there are no attributes that are dependent on a non-primary key.

**IP ITEM** (IP\_Item\_ID, Store\_ID (FK), Title, IP\_Description, Price, File\_Type)

Store ID is a foreign key originating from **VIRTUAL STORE**.

IP\_Item’s functional dependencies are the following:

{IP\_Item\_ID} -> {Title, IP\_Description, Price, File\_Type, Store\_ID}

It is in 1NF because it meets all four requirements, including that all values are atomic.

It is 2NF because there are no partial dependencies, as there is only one determinant.

It is 3NF because there are no transitive dependencies, as there is only one determinant.

It is BCNF because there are no attributes that are dependent on a non-primary key.

**IP ITEM IMAGES** (Image\_ID, IP\_Item\_ID (FK), Image\_Description, Image\_Name, File\_Type, Upload\_Date)

IP Item ID is a foreign key originating from **IP ITEM**.

IP\_Item\_Images’ functional dependencies are the following:

{Image\_ID} -> {Image\_Description, Image\_Name, File\_Type, Upload\_Date, IP\_Item\_ID}

It is in 1NF because it meets all four requirements, including that all values are atomic.

It is 2NF because there are no partial dependencies, as there is only one determinant.

It is 3NF because there are no transitive dependencies, as there is only one determinant.

It is BCNF because there are no attributes that are dependent on a non-primary key.

**IP ITEM KEYWORDS** (Keyword\_ID, IP\_Item\_ID (FK), Keyword\_Name, Upload\_Date)

IP Item ID is a foriegn key originating from **IP ITEM**.

IP\_Item\_Keywords’ functional dependencies are the following:

{Keyword\_ID} -> {Keyword\_Name, Upload\_Date, IP\_Item\_ID}

It is in 1NF because it meets all four requirements, including that all values are atomic.

It is 2NF because there are no partial dependencies, as there is only one determinant.

It is 3NF because there are no transitive dependencies, as there is only one determinant.

It is BCNF because there are no attributes that are dependent on a non-primary key.

**VIRTUAL STORE** (Store ID, User\_ID (FK), Store\_Description, Store\_Banner, Name, Store\_Photo, Store\_Bio, Store\_Category ID (FK))

User ID is a foreign key originating from **USER**.

Store\_Category is a foreign key originating from Category Name in **STORE CATEGORIES.**

Virtual Store’s functional dependencies are the following:

{Store\_ID} -> {Store\_Description, Store\_Banner, Name, Store\_Photo, Store\_Bio, User\_ID, Store\_Category\_ID}

It is in 1NF because it meets all four requirements, including that all values are atomic.

It is 2NF because there are no partial dependencies, as there is only one determinant.

It is 3NF because there are no transitive dependencies, as there is only one determinant.

It is BCNF because there are no attributes that are dependent on a non-primary key.

**VIRTUAL STORE URLS** (URL\_ID, Store\_ID (FK), URL\_Name, URL\_Link, Upload\_Date)

Store ID is a foreign key originating from **VIRTUAL STORE**.

Virtual Store URLS’ functional dependencies are the following:

{URL\_ID} -> {URL\_Name, URL\_Link, Upload\_Date, Store\_ID}

It is in 1NF because it meets all four requirements, including that all values are atomic.

It is 2NF because there are no partial dependencies, as there is only one determinant.

It is 3NF because there are no transitive dependencies, as there is only one determinant.

It is BCNF because there are no attributes that are dependent on a non-primary key.

**IP ITEM REVIEWED** (IP Item ID, Review ID)

IP Item ID is a foreign key originating from **IP ITEM**

Review ID is a foregin key originating from **REVIEW**

IP Item Reviewed’s functional dependencies are the following:

{IP Item ID, Review ID}

It is in 1NF because it meets all four requirements, including that all values are atomic.

It is 2NF because there are no partial dependencies.

It is 3NF because there are no transitive dependencies.

It is BCNF because there are no attributes that are dependent on a non-primary key.

**ORDER REVIEWED** (Order ID, Review ID)

Order ID is a foreign key originating from **ORDER**

Review ID is a foreign key originating from **REVIEW**

Order Reviewed’s functional dependencies are the following:

{Order ID, Review ID}

It is in 1NF because it meets all four requirements, including that all values are atomic.

It is 2NF because there are no partial dependencies.

It is 3NF because there are no transitive dependencies.

It is BCNF because there are no attributes that are dependent on a non-primary key.

**STORE CATEGORIES** (Store\_Category ID, Category\_Name, Description)

Store ID is a foreign key originating from **VIRTUAL STORE**.

Store Categories’ functional dependencies are the following:

{Store\_Category\_ID} -> {Category\_Name, Description}

{Category\_Name} -> {Store\_Category\_ID, Description}

It is in 1NF because it meets all four requirements, including that all values are atomic.

It is 2NF because there are no partial dependencies, as there is only one determinant.

It is 3NF because there are no transitive dependencies, as there is only one determinant.

It is BCNF because there are no attributes that are dependent on a non-primary key. Category\_Name is a candidate key.

**2a Table Description**

The following are our DB tables. Before showing our table, we will provide some information about it.

**Below is our User table. It stores crucial information about each user, including flagging them as a buyer or seller depending on the value of buyer\_flag and seller\_flag. There are no foreign keys, but User\_ID is used a foreign key in many other tables.**

CREATE TABLE USER

( User\_ID INT NOT NULL,

First\_Name VARCHAR(15) NOT NULL,

Last\_Name VARCHAR(15) NOT NULL,

Password VARCHAR(15) NOT NULL,

Buyer\_Flag TINYINT NOT NULL,

Seller\_Flag TINYINT NOT NULL,

Email VARCHAR(15) NOT NULL,

CONSTRAINT UPK

PRIMARY KEY(User\_ID));

**Below is our IP\_Item table. This table defines an item within the requirements, such as allowing the user to specify the title, description, price, and file type. This table has a foreign key, Store\_ID, which represents which store the item will be listed on.**

CREATE TABLE IP\_ITEM

( IP\_Item\_ID INT NOT NULL,

Store\_ID INT NOT NULL,

Title VARCHAR(15) NOT NULL,

IP\_Description VARCHAR(15),

Price REAL NOT NULL,

File\_Type INT NOT NULL,

CONSTRAINT IIPK

PRIMARY KEY(IP\_Item\_ID),

CONSTRAINT IISIFK

FOREIGN KEY(Store\_ID) REFERENCES VIRTUAL\_STORE(Store\_ID));

**Below is our IP\_Item\_Images table. As part of the requirements for our IP\_Item, we must be able to associate images with IP\_Items. This table serves that purpose, and is related to the item it holds pictures for through the IP\_Item\_ID foreign key.**

CREATE TABLE IP\_ITEM\_IMAGES

( Image\_ID INT NOT NULL,

IP\_Item\_ID INT NOT NULL,

Image\_Description VARCHAR(15) NOT NULL,

Image\_Name VARCHAR(15) NOT NULL,

File\_Type VARCHAR(15) NOT NULL,

Upload\_Date DATE NOT NULL,

CONSTRAINT IIIPK

PRIMARY KEY(Image\_ID),

CONSTRAINT IIIIIFK

FOREIGN KEY(IP\_Item\_ID) REFERENCES VIRTUAL\_STORE(IP\_Item\_ID));

**Below is our IP\_Item\_Keywords table. This fulfills another requirement for IP Items, which is to have keywords for each item. Each keyword is stored here, as well as the upload date of the keyword. It is linked to the IP\_Item through the IP\_Item\_ID foreign key.**

CREATE TABLE IP\_ITEM\_KEYWORDS

( Keyword\_ID INT NOT NULL,

IP\_Item\_ID INT NOT NULL,

Keyword\_Name VARCHAR(15) NOT NULL,

Upload\_Date DATE NOT NULL,

CONSTRAINT IIKKIPK

PRIMARY KEY(Keyword\_ID),

CONSTRAINT IIKIIIFK

FOREIGN KEY(IP\_Item\_ID) REFERENCES IP\_ITEM(IP\_Item\_ID));

**Below is our Items\_In\_Wishlist table. This table represents the N:M relationship between a user’s wishlist, and the items that are put onto that wishlist. Each record consists of an item that is on a wishlist, and a specific wishlist it is on. There are no new attributes for this table, it simply uses the Wishlist\_ID from Wishlist and the IP\_Item\_ID from IP\_Item as foreign keys.**

CREATE TABLE ITEMS\_IN\_WISHLIST

( Wishlist\_ID INT NOT NULL,

IP\_Item\_ID INT NOT NULL DEFAULT 0,

CONSTRAINT IIWPK

PRIMARY KEY(Wishlist\_ID),

CONSTRAINT IIWWIFK

FOREIGN KEY(Wishlist\_ID) REFERENCES WISHLIST(Wishlist\_ID)

ON DELETE SET DEFAULT ON UPDATE CASCADE,

CONSTRAINT IIWIPFK

FOREIGN KEY(IP\_Item\_ID) REFERENCES IP\_ITEM(IP\_Item\_ID)

ON DELETE SET DEFAULT ON UPDATE CASCADE);

**Below is our Wishlist table. This is another simple table, it links a buyer’s User\_ID to a Wishlist\_ID. Each buyer can only have one wishlist.**

CREATE TABLE WISHLIST

( Wishlist\_ID INT NOT NULL,

User\_ID INT NOT NULL DEFAULT 0,

CONSTRAINT WPK

PRIMARY KEY(Wishlist\_ID)

CONSTRAINT WFK

FOREIGN KEY(User\_ID) REFERENCES USER(User\_ID)

ON DELETE SET DEFAULT ON UPDATE CASCADE);

**Below is our Item\_Added\_To\_Order table. This table represents the N:M relationship between Item and Order. Each item that is ordered will have a record here with its corresponding Order\_ID, which represents the order that the item was purchased in. The quantity of the item purchased is recorded in the Quantity attribute.**

CREATE TABLE ITEM\_ADDED\_TO\_ORDER

( Order\_ID INT NOT NULL,

IP\_Item\_ID INT NOT NULL DEFAULT 0,

Quantity INT NOT NULL,

CONSTRAINT IATOPK

PRIMARY KEY(Order\_ID),

CONSTRAINT IATOOIFK

FOREIGN KEY(Order\_ID) REFERENCES ITEM\_ORDER(Order\_ID),

CONSTRAINT IATOIIIFK

FOREIGN KEY(IP\_Item\_ID) REFERENCES IP\_ITEM(IP\_Item\_ID)

ON DELETE SET DEFAULT ON UPDATE CASCADE);

**Below is our Item\_Order table. This table represents an order placed by a user. The date of the order is recorded in Order\_Date. The user who placed the order is listed in the foreign key User\_ID.**

CREATE TABLE ITEM\_ORDER

( Order\_ID INT NOT NULL,

Order\_Date DATE NOT NULL,

Price REAL NOT NULL,

User\_ID INT NOT NULL,

CONSTRAINT IOPK

PRIMARY KEY(Order\_ID),

CONSTRAINT IOFK

FOREIGN KEY(Price) REFERENCES IP\_ITEM(Price),

CONSTRAINT IOUIFK

FOREIGN KEY(User\_ID) REFERENCES USER(User\_ID));

**Below is our IP\_Item\_Reviewed table. This table is created for the N:M relationship between IP\_Item and Review. For each review made, it will store the Review\_ID, as well as the IP\_Item\_ID of the item the review is for, both of the previous attributes are foreign keys.**

CREATE TABLE IP\_ITEM\_REVIEWED

( Review\_ID INT NOT NULL,

IP\_Item\_ID INT NOT NULL,

CONSTRAINT IRPK

PRIMARY KEY(Review\_ID),

CONSTRAINT IRRIFK

FOREIGN KEY(Review\_ID) REFERENCES REVIEW(Review\_ID),

CONSTRAINT IROIFK

FOREIGN KEY(IP\_Item\_ID) REFERENCES IP\_ITEM(IP\_Item\_ID));

**Below is our Order\_Reviewed table. This table is created for the N:M relationship between Review and Order. For each review made, it swill store the Order\_ID of the order the review is originating from, as well as the Review\_ID linking to the review. This table helps ensure that only users who purchased the item in an order will be able to leave a review.**

CREATE TABLE ORDER\_REVIEWED

( Review\_ID INT NOT NULL,

Order\_ID INT NOT NULL,

CONSTRAINT IRPK

PRIMARY KEY(Review\_ID),

CONSTRAINT IRRIFK

FOREIGN KEY(Review\_ID) REFERENCES REVIEW(Review\_ID),

CONSTRAINT IROIFK

FOREIGN KEY(Order\_ID) REFERENCES ITEM\_ORDER(Order\_ID));

**Below is our Review table. This table consists of various attributes that relate to the reviews that user’s can leave on IP\_Items, assuming that they have purchased the item in an order.**

CREATE TABLE REVIEW

( Review\_ID INT NOT NULL,

Rating INT NOT NULL,

Description VARCHAR(2000),

IP\_Item\_ID INT NOT NULL DEFAULT 0,

User\_ID INT NOT NULL DEFAULT 0,

CONSTRAINT RPK

PRIMARY KEY(Review\_ID),

CONSTRAINT RIPFK

FOREIGN KEY(IP\_Item\_ID) REFERENCES IP\_ITEM(IP\_Item\_ID)

ON DELETE SET DEFAULT ON UPDATE CASCADE,

CONSTRAINT RUIFK

FOREIGN KEY(User\_ID) REFERENCES USER(User\_ID)

ON DELETE SET DEFAULT ON UPDATE CASCADE);

**Below is our Virtual Store table. This table stores the virtual stores that can be created by sellers. Various attributes can be set, such as the name of the store, a store description, a store banner, and more. The table is linked to the seller that owns the store through the User\_ID foreign key. The virtual store will be related to its store category through a foreign key with Store\_Category**

CREATE TABLE VIRTUAL\_STORE

( Store\_ID INT NOT NULL,

User\_ID INT NOT NULL,

Store\_Description VARCHAR(15),

Store\_Banner VARCHAR(15),

Name VARCHAR(15) NOT NULL,

Seller\_Bio VARCHAR(15),

Store\_Category VARCHAR(15),

CONSTRAINT VSSIPK

PRIMARY KEY(Store\_ID),

CONSTRAINT VSUIFK

FOREIGN KEY(User\_ID) REFERENCES USER(User\_ID));

CONSTAINT VSCIFK

FOREIGN KEY(Store\_Category\_ID) REFERENCES STORE\_CATEGORY(Store\_Category\_ID)

**Below is our Virtual Store URLs table. As part of the virtual store requirement, URLs should be able to be listed for each virtual store. This table will store these links, as well as some information about them, and then relate it to the virtual store through the Store\_ID foreign key.**

CREATE TABLE VIRTUAL\_STORE\_URLS

( URL\_ID INT NOT NULL,

URL\_Name VARCHAR(15) NOT NULL,

URL\_Link VARCHAR(15) NOT NULL,

Store\_ID INT NOT NULL,

Upload\_Date DATE NOT NULL,

CONSTRAINT VSUUIPK

PRIMARY KEY(URL\_ID),

CONSTRAINT VSUSIFK

FOREIGN KEY(Store\_ID) REFERENCES VIRTUAL\_STORE(Store\_ID));

**Below is our Store Categories table. This table contains our store categories, as well as some information about them, contained in the description attribute. The Store\_Category\_ID will be used as a foreign key in virtual stores to designate which stores should be put into which categories.**

CREATE TABLE STORE\_CATEGORIES

( Store\_Category\_ID INT NOT NULL,

Category\_Name VARCHAR(15) NOT NULL,

Description VARCHAR(15),

CONSTRAINT SCCNPK

PRIMARY KEY(Store\_Category\_ID),

**Below is our Charges table. This table represents the N:M relationship between Item Order and Payment Methods. This table is used to designate the amount of money paid towards an order using a particular payment method. This table is necessary since the buyer will be able to use multiple payment methods on a single purchase, as stated in the requirements.**

CREATE TABLE CHARGES

( Charge\_ID INT NOT NULL,

Payment\_Method\_ID INT NOT NULL,

Charge\_Amount REAL NOT NULL,

CONSTRAINT CCIPK

PRIMARY KEY(Charge\_ID),

CONSTRAINT CCIFK

FOREIGN KEY(Charge\_ID) REFERENCES PAYMENT\_METHODS(Charge\_ID),

CONSTRAINT CPMIFK

FOREIGN KEY(Payment\_Method\_ID) REFERENCES PAYMENT\_METHODS(Payment\_Method\_ID));

**Below is our Payment Methods table. This table contains our Payment\_Method\_ID, which is related to each of the payment methods that buyers can use to make purchases on the store.**

CREATE TABLE PAYMENT\_METHODS

( Payment\_Method\_ID INT NOT NULL,

Preferred\_Payment TINYINT NOT NULL,

Charge\_ID INT NOT NULL,

CONSTRAINT PMPK

PRIMARY KEY(Payment\_Method\_ID));

**Below is our Bank Account table. This table contains all of the information needed to make a purchase with the user’s bank account, such as the account holder name and the account number. It contains a foreign key of Payment\_Method\_ID that relates this to Payment Methods.**

CREATE TABLE BANK\_ACCOUNT

( Payment\_Method\_ID INT NOT NULL,

Account\_Holder\_Name VARCHAR(15) NOT NULL,

Account\_Number INT NOT NULL,

Charge\_ID INT NOT NULL,

User\_ID INT NOT NULL DEFAULT 0,

CONSTRAINT BAPK

PRIMARY KEY(Payment\_Method\_ID),

CONSTRAINT CCCIFK

FOREIGN KEY(Charge\_ID) REFERENCES PAYMENT\_METHODS(Charge\_ID),

CONSTRAINT BAUIFK

FOREIGN KEY(User\_ID) references USER(User\_ID)

ON DELETE SET DEFAULT ON UPDATE CASCADE);

**Below is our Credit Card table. This table contains all of the information needed to make a purchase with the user’s credit card, such as the card number and the CVV. It contains a foreign key of Payment\_Method\_ID that relates this to Payment Methods.**

CREATE TABLE CREDIT\_CARD

( Payment\_Method\_ID INT NOT NULL,

CVV INT NOT NULL,

Card\_Number INT NOT NULL,

Cardholder\_Name VARCHAR(15) NOT NULL,

Charge\_ID INT NOT NULL,

User\_ID INT NOT NULL,

CONSTRAINT CCPK

PRIMARY KEY(Payment\_Method\_ID),

CONSTRAINT CCCIFK

FOREIGN KEY(Charge\_ID) REFERENCES PAYMENT\_METHODS(Charge\_ID),

CONSTRAINT CCUIFK

FOREIGN KEY(User\_ID) REFERENCES USER(User\_ID),

CONSTRAINT CCPMIFK

FOREIGN KEY(Payment\_Method\_ID) REFERENCES PAYMENT\_METHODS(Payment\_Method\_ID)

ON DELETE SET DEFAULT ON UPDATE CASCADE);

**Below is our Karma table. This table contains all of the information needed to make a purchase with the user’s bank account, such as the karma wallet ID and the user’s karma balance. It contains a foreign key of Payment\_Method\_ID that relates this to Payment Methods.**

CREATE TABLE KARMA

( Payment\_Method\_ID INT NOT NULL,

Karma\_Wallet\_ID INT NOT NULL,

Karma\_Balance INT NOT NULL,

Charge\_ID INT NOT NULL,

User\_ID INT NOT NULL DEFAULT 0,

CONSTRAINT KPK

PRIMARY KEY(Payment\_Method\_ID),

CONSTRAINT CCCIFK

FOREIGN KEY(Charge\_ID) REFERENCES PAYMENT\_METHODS(Charge\_ID),

CONSTRAINT KUIFK

FOREIGN KEY(User\_ID) REFERENCES USER(User\_ID)

ON DELETE SET DEFAULT ON UPDATE CASCADE,

CONSTRAINT KPMIFK

FOREIGN KEY(Payment\_Method\_ID) REFERENCES PAYMENT\_METHODS(Payment\_Method\_ID)

ON DELETE SET DEFAULT ON UPDATE CASCADE);

**Below is our Crypto table. This table contains all of the information needed to make a purchase with the user’s crypto, such as the user’s crypto wallet id. It contains a foreign key of Payment\_Method\_ID that relates this to Payment Methods.**

CREATE TABLE CRYPTO

( Payment\_Method\_ID INT NOT NULL,

Crypto\_Wallet\_ID INT NOT NULL,

Charge\_ID INT NOT NULL,

User\_ID INT NOT NULL,

CONSTRAINT CPK

PRIMARY KEY(Payment\_Method\_ID),

CONSTRAINT CPMIFK

FOREIGN KEY(Payment\_Method\_ID) REFERENCES PAYMENT\_METHODS(Payment\_Method\_ID),

CONSTRAINT CCCIFK

FOREIGN KEY(Charge\_ID) REFERENCES PAYMENT\_METHODS(Charge\_ID),

CONSTRAINT CUIFK

FOREIGN KEY(User\_ID) REFERENCES USER(User\_ID));

**2b Catalog of SQL Queries**

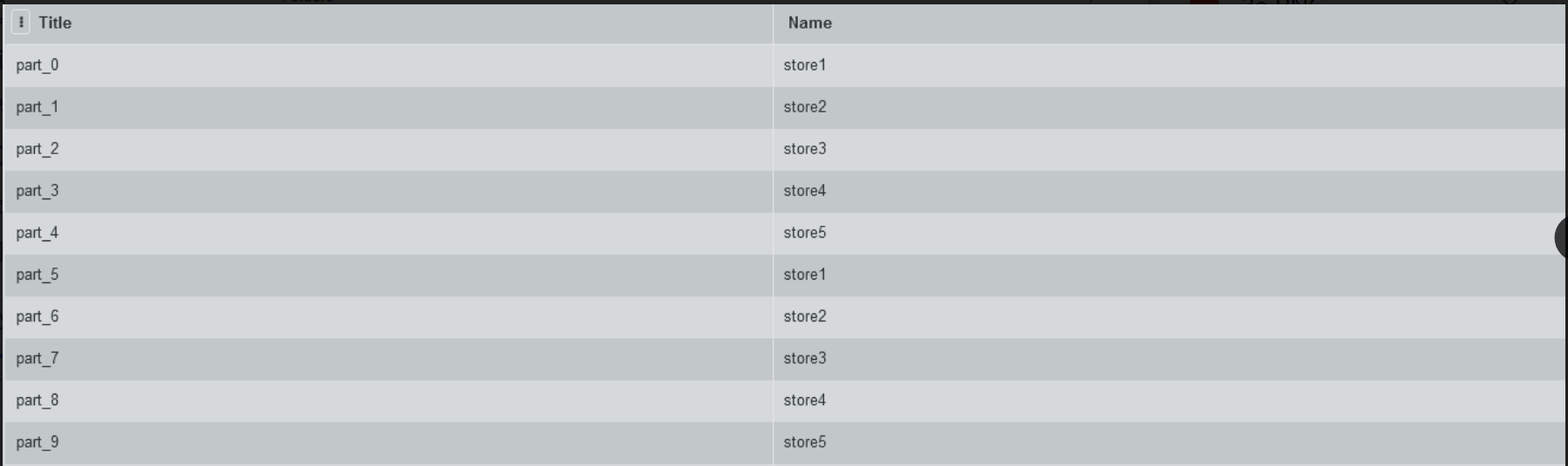
The following is our catalog of SQL queries. We will have the query, and then explain it.

3a Create a list of IP items and the stores selling those

SELECT IP.Title, VS.Name

FROM IP\_ITEM AS IP, VIRTUAL\_STORE AS VS

WHERE IP.Store\_ID == VS.Store\_ID;



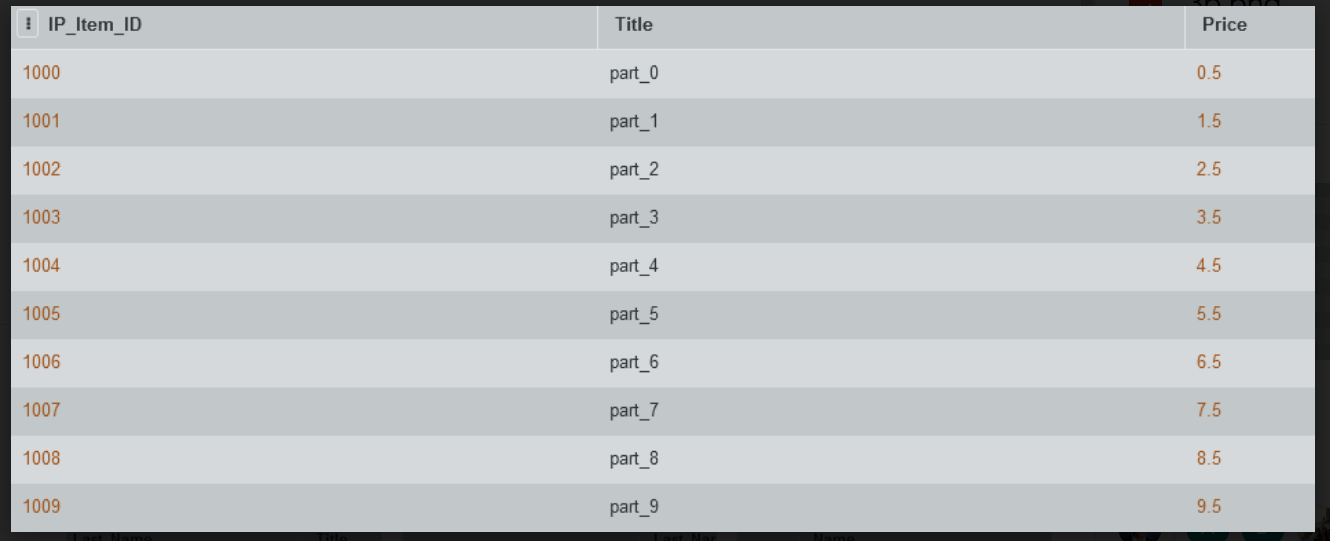
**Above is a query that will list the name of all of the items in our database in the first column, and the corresponding store that sells the item in the second column.**

3b Find the titles of all IP Items that cost less than $10.

SELECT IP.Title

FROM IP\_ITEM AS IP

WHERE IP.Price < 10;



**Above is a query that will look at all items in our database, and then list the items with a price of less than 10 in the first and only column of this query.**

3c Generate a list of IP item titles and dates of purchase made by a given buyer (you choose how to designate a buyer).

SELECT IP.Title, IO.Order\_Date

FROM BUYER AS B, ITEM\_ORDER AS IO, ITEM\_ADDED\_TO\_ORDER AS IATO, IP\_ITEM AS IP

WHERE B.User\_ID = ‘0000’

AND B.User\_ID = IO.User\_ID

AND IO.Order\_ID = IATO.Order\_ID

AND IATO.IP\_Item\_ID = IP.IP\_Item\_ID;



**Above is a query that will take a specified User ID corresponding to a buyer, find all the orders that the user has placed, find all the items that a user has purchased in an order, and then find the details needed for all of the items that the user has purchased.**

3d List all the buyers who purchased an IP Item from a given store (you choose how to designate a store) and the names of the IP Items they purchased.

SELECT VS.Name, B.First\_Name, B.Last\_Name, IP.Title

FROM BUYER AS B, ITEM\_ORDER AS IO, ITEM\_ADDED\_TO\_ORDER AS IATO, IP\_ITEM AS IP, VIRTUAL\_STORE AS VS

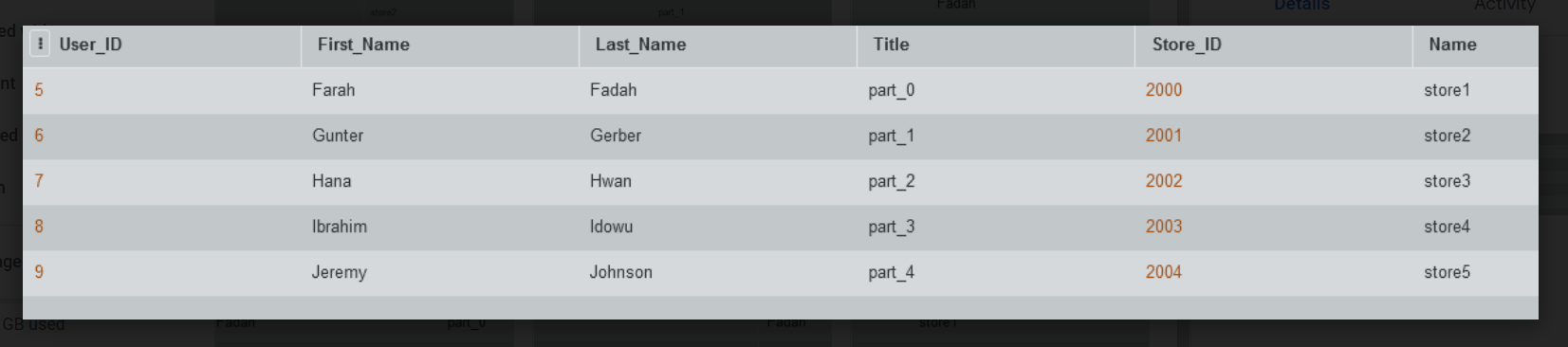
WHERE VS.Store\_ID = ‘2000’

AND VS.Store\_ID = IP.Store\_ID

AND IP.IP\_Item\_ID = IATO.IP\_Item\_ID

AND IATO.Order\_ID = IO.Order\_ID

AND IO.User\_ID = B.User\_ID;



**Above is a query that will take a specified Virtual\_Store\_ID, find the IP\_Items listed on the store, the orders where those items have been purchased, and then the user who placed those orders. We will then have a query that includes the name of the store, the name of the buyers, and the names of the items they have purchased.**

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

SELECT B.First\_Name, B.Last\_Name, COUNT(IP\_Item.ID)

FROM BUYER AS B, ITEM\_ORDER AS IO, ITEM\_ADDED\_TO\_ORDER AS IATO

WHERE B.User\_ID = IO.User\_ID

AND IO.Order\_ID = IATO.Order\_ID

AND MAX(IP\_Item.ID)

GROUP BY B.User\_ID;



3f Create a list of stores who currently offer 5 or less IP Items for sale

SELECT VS.Name

FROM VIRTUAL\_STORE AS VS, IP\_ITEM AS IP

WHERE VS.Store\_ID = IP.Store\_ID

GROUP BY VS.Store\_ID

HAVING COUNT(VS.Store\_ID) < 6



**Above is a query that will join all of the virtual stores with all of the items that they sell. Then, we will count the instances of the same Store\_ID, which will tell us how many products there are per store. We then use HAVING to only list those stores where the number of products is less than 6.**

3g 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.

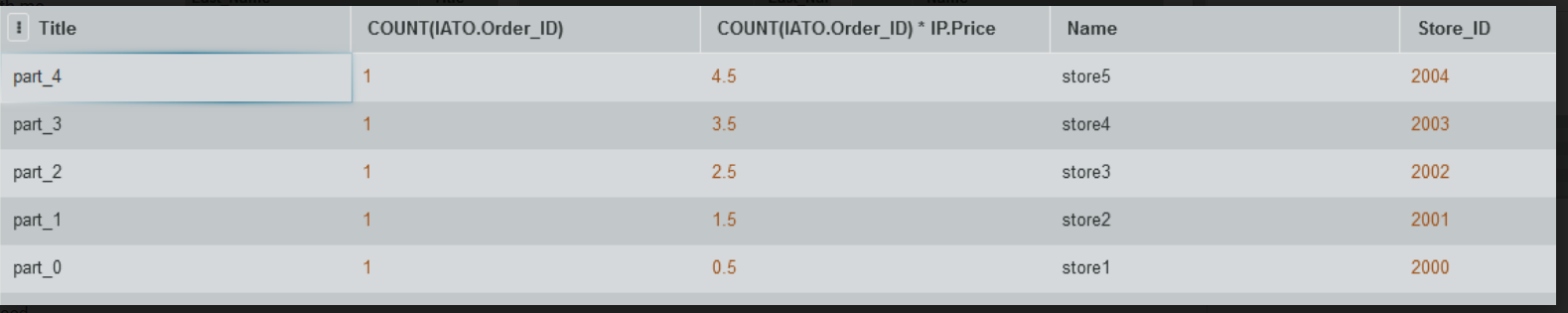
SELECT IP.Title, revenue, units\_sold, Store\_ID FROM

(SELECT IP.Title, COUNT(Order\_ID)\*IP.Price AS revenue, COUNT(Order\_ID) as units\_sold, Store\_ID FROM IP\_ITEM AS IP, ITEM\_ADDED\_TO\_ORDER AS IATO

WHERE IP.IP\_Item\_ID = IATO.IP\_Item\_ID

GROUP BY IP.IP\_Item\_ID

ORDER BY revenue DESC);



3h 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.

SELECT C.Payment\_Method\_Type, COUNT(C.Payment\_Method\_Type), SUM(IP.Price)

FROM CHARGES AS C, ITEM\_ORDER AS IO, ITEM\_ADDED\_TO\_ORDER AS IATO, IP\_ITEM AS IP

WHERE C.Order\_ID = IO.Order\_ID

AND IO.Order\_ID = IATO.Order\_ID  
AND IATO.IP\_Item\_ID = IP.IP\_Item\_ID

GROUP BY C.Payment\_Method\_Type;



**Above is a query that will take all of the charges made and count them based on the payment type made. After getting that, we then join with orders, the item added to order, and the item itself so we can calculate the total price spent using each payment method.**

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

SELECT B.First\_Name, B.Last\_Name, B.Email

FROM KARMA AS K, BUYER AS B

WHERE Karma.User\_ID = User.User\_ID

HAVING MAX(Karma\_Balance);



**We will join all users with their karma balance, find the highest karma value, and then display that user’s information.**

4a How many stores are in each category

SELECT SC.Category\_Name, COUNT(VS.Store\_ID)

FROM STORE\_CATEGORIES AS SC, VIRTUAL\_STORE AS VS

WHERE SC.Store\_Category\_ID = VS.Store\_Category\_ID

GROUP BY Store\_ID;

4b Find how many wishlists each IP Item is on

SELECT IIW.IP\_Item\_Name, COUNT(IIW.IP\_Item\_ID)

FROM ITEMS\_IN\_WISHLIST AS IIW

GROUP BY IIW.IP\_Item\_Name;

4c Average cost of item on each store

SELECT VS.Name, AVG(IP.Price)

FROM VIRTUAL\_STORE AS VS, IP\_ITEM AS IP

WHERE IP.Store\_ID = VS.Store\_ID

GROUP BY VS.Store\_ID;

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

SELECT B.First\_Name, B.Last\_Name, SUM(IP.Price)

FROM BUYER AS B, ITEM\_ORDER AS IO, ITEM\_ADDED\_TO\_ORDER AS IATO, IP\_ITEM AS IP

WHERE B.User\_ID = IO.User\_ID

AND IO.Order\_ID = IATO.Order\_ID

AND IATO.IP\_Item\_ID = IP.IP\_Item\_ID

GROUP BY B.User\_ID;

**The above query will take each buyer, join their records with all of the items they have placed order for, and then a sum of the price of all the items, using the User\_ID to group, as well as selecting the buyer’s name to display.**

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

SELECT B.First\_Name, B.Last\_Name, B.Email

FROM BUYER AS B, ITEM\_ORDER AS IO, ITEM\_ADDED\_TO\_ORDER AS IATO, IP\_ITEM AS IP

WHERE B.User\_ID = IO.User\_ID

AND IO.Order\_ID = IATO.Order\_ID

AND IATO.IP\_Item\_ID = IP.IP\_Item\_ID

GROUP BY B.User\_ID

HAVING SUM(IP.Price) >

(SELECT AVG(total\_price)

FROM (SELECT SUM(IP.Price) AS total\_price FROM USER AS B, ITEM\_ORDER AS IO, ITEM\_ADDED\_TO\_ORDER AS IATO, IP\_ITEM AS IP

WHERE B.User\_ID = IO.User\_ID

AND IO.Order\_ID = IATO.Order\_ID

AND IATO.IP\_Item\_ID = IP.IP\_Item\_ID

GROUP BY B.User\_ID));

**The above query will find the sum that all buyers have spent on items, and then select the first name, last name, and email address for all the buyers who have spent higher than the average buyer. This is done by calculating the average amount spent by each buyer and then comparing the amount each buyer has spent to that average value.**

5c

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 IP.Title, COUNT(IP.IP\_Item\_ID)

FROM IP\_ITEM AS IP, ITEM\_ADDED\_TO\_ORDER AS IATO

WHERE IP.IP\_Item\_ID = IATO.IP\_Item\_ID

GROUP BY IP.IP\_Item\_ID

ORDER BY COUNT(IP.IP\_Item\_ID) DESC;

**The above query will count the number of times an item has been ordered by counting the number of occurrences of each IP\_Item\_ID in the joined table of IP\_Item and Item\_Added\_To\_Order. Upon counting, our query will sort the count from most to least by using ORDER BY.**

5d 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 IP.Title, COUNT(Order ID)\*IP.Price AS revenue

FROM IP\_ITEM AS IP, ITEM\_ADDED\_TO\_ORDER AS IATO

WHERE IP.IP\_Item\_ID = IATO.IP\_Item\_ID

GROUP BY IP.IP\_Item\_ID

ORDER BY revenue DESC;

**The above query will join IP\_Item and Item\_Added\_To\_Order to create a complete record of each time an item has been purchased on our site. We will then group by the IP\_Item\_ID. We then count the number of instances of the IP\_Item\_ID, which tells us how many times the item has been sold, and multiply that value by the price of the IP item. We then sort revenue from highest to lowest by using ORDER BY.**

5e Find the most popular seller (i.e. the one who has sold the most IP Items)

SELECT S.First\_Name, S.Last\_Name, MAX(items\_sold)

FROM (SELECT S.First\_Name, S.Last\_Name, COUNT(IP\_Item\_ID) AS items\_sold FROM ITEM\_ADDED\_TO\_ORDER AS IATO, IP\_ITEM AS IP, VIRTUAL\_STORE AS VS, USER AS S

WHERE IATO.IP\_Item\_ID = IP.IP\_Item\_ID

AND IP.Store\_ID = VS.Store\_ID

AND VS.User\_ID = S.User\_ID

GROUP BY S.User\_ID);

**The above query will use Item\_Added\_To\_Order to list all of the times an item has been purchased on our site. We will count the number of IP\_Item\_ID’s to get a count of the items sold and then group by the seller’s User\_ID. Upon doing all of this, we can take the max of the items sold by each of our sellers, and we will have our most popular seller.**

5f Find the most profitable seller (i.e. the one who has brought in the most money)

SELECT S.First\_Name, S.Last\_Name, MAX(profit)

FROM (SELECT S.First\_Name, S.Last\_Name, SUM(IP.Price) AS profit FROM ITEM\_ADDED\_TO\_ORDER AS IATO, IP\_ITEM AS IP, VIRTUAL\_STORE AS VS, USER AS S

WHERE IATO.IP\_Item\_ID = IP.IP\_Item\_ID

AND IP.Store\_ID = VS.Store\_ID

AND VS.User\_ID = S.User\_ID

GROUP BY S.User\_ID);

**The above query uses Item\_Added\_To\_Order to find each instance of an item being sold. We then sum the prices of each of the items, grouping by the seller’s User\_ID. This will give us all of our sellers and the profit they have made. When then take the max of the profit generated by each of our sellers to find our most profitable seller.**

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

SELECT B.First\_Name, B.Last\_Name

FROM User as B

WHERE Order\_ID IN

SELECT S.User\_ID, MAX(profit)

FROM (SELECT S.First\_Name, S.Last\_Name, SUM(IP.Price) AS profit FROM ITEM\_ADDED\_TO\_ORDER AS IATO, IP\_ITEM AS IP, VIRTUAL\_STORE AS VS, USER AS S

WHERE IATO.IP\_Item\_ID = IP.IP\_Item\_ID

AND IP.Store\_ID = VS.Store\_ID

AND VS.User\_ID = S.User\_ID

GROUP BY S.User\_ID);

**The above query will first find the most profitable seller in a similar way to 5g, by grouping the items purchased in every order together by the seller’s User\_ID and finding the sum. Upon taking the max value from that, and also storing the User\_ID of the most profitable seller.**

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

5i 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 VS.Name, COUNT(IP.IP\_Item\_ID), MAX(IP.Price), MIN(IP.Price), AVG(IP.Price)

FROM VIRTUAL\_STORE AS VS, IP\_ITEM AS IP

WHERE VS.Store\_ID = ‘2000’

AND VS.Store\_ID = IP.Store\_ID

**The above query will look at each IP Item that is listed on the store by using a join.**

**2c INSERT and DELETE Queries**

For our insert queries, we will enter a new seller into our site, create a new store with that new user as the owner, and then create a new item to insert into our store.

INSERT INTO USER (User\_ID, First\_Name, Last\_Name, Password, Buyer\_Flag, Seller\_Flag, Email)

VALUES ('0010', 'Seller', 'Example', 'password10', '0', '1', 'email10.com');

INSERT INTO VIRTUAL\_STORE (Store\_ID, User\_ID, Store\_Description, Store\_Banner, Name, Seller\_Bio, Store\_Category)

VALUES ('2010', '0010', 'description10', 'banner10', 'store10', 'bio10', 'category1');

INSERT INTO IP\_ITEM (IP\_Item\_ID, Store\_ID, Title, IP\_Description, Price, File\_Type)

VALUES ('1010', '2010', 'part\_10', 'description\_10', '2.50', '.stl');

For our delete queries, we will delete a seller from our site. We will do this by first deleting the items they have listed on the site. We will then delete the store they have listed on the site. Finally, we will delete the user from the site.

DELETE FROM IP\_ITEM

WHERE IP\_Item\_ID = ‘1010’;

DELETE FROM VIRTUAL\_STORE

WHERE Store\_ID = ‘2010’;

DELETE FROM USER

WHERE User\_ID = ‘0010’;

**2d Indexes**

Our first index will be on the Price attribute of our IP\_Item entity. This attribute is used in many cases, such as finding items that are greater than, less than, or between a certain price. We will use a B-tree index, since most of our queries involving the price of an IP\_Item will be range tests.

CREATE UNIQUE INDEX IP\_ITEM\_PRICE

ON IP\_ITEM (Price ASC);

Our second index will be on the Name attribute of our Virtual\_Store entity. We will need the name of a store in many cases, such as when we are searching for stores that have certain qualities, like being in a specific store category or doing a certain amount of sales. We will use a Hash index, since most of our queries will be searching for the exact name of the store, and we will do very few range tests on this index.

CREATE UNIQUE INDEX STORE\_NAME

ON VIRTUAL\_STORE (Name ASC);

**2e Views**

This table will create a table that lists each store that has at least one item on sale for less than the average cost of all of the items in our database.

CREATE VIEW StoresItemLessAverage

AS

SELECT VS.Name

FROM VIRTUAL\_STORE as VS, IP\_ITEM as IP

WHERE IP.Price < AVG(IP.Price)

AND VS.Store\_ID = IP.Store\_ID

This view will create a table that gives the sum of each user’s wishlist.

CREATE VIEW WishlistSum

AS

SELECT B.First\_Name, B.Last\_Name, SUM(IP.Price))

FROM USER as B, WISHLIST as W, ITEMS\_IN\_WISHLIST as IIW, IP\_ITEM as IP

WHERE B.User\_ID = W.User\_ID

AND W.Wishlist\_ID = IIW.Wishlist\_ID

AND IIW.IP\_Item\_ID = IP.IP\_Item\_ID

GROUP BY B.User\_ID;

**2f Transactions**

**The COMPLETE\_PURCHASE transaction is very important. It ensures that the buyer must have sufficient funds to complete a purchase and be charged correctly. If the buyer does not have the funds the transaction would be canceled. This is to ensure that if a seller completes a sale they will receive proper compensation for their product. For this example, we decided to use Karma to track if the buyer has proper funds. If there is an issue with the Karma balance we will decide to immediately cancel the order. The transaction goes as follows:**

BEGIN TRANSACTION COMPLETE\_PURCHASE

INSERT INTO KARMA ('11000', '12000', '1000', '10000', '0000');

IF error THEN GO TO UNDO; END IF;

UPDATE Payment\_Method\_ID SET Karma\_Balance = Karma\_Balance + 1000

WHERE Payment\_Method\_ID == 11000;

IF error THEN GO TO UNDO; END IF;

COMMIT;

GO TO FINISH;

UNDO:

ROLLBACK;

FINISH:

END TRANSACTION;

**We created the SUS\_ALERT transaction to help suspend accounts for suspicious activity. If a buyer’s Karma balance has reached a certain level, in this instance 10,000, we will then suspend them from any buyer actions, doing nothing to seller actions. This is to make sure nothing suspicious is occurring, such as someone giving themselves so much Karma somehow. This basically protects the service’s Karma currency from being misused and taken advantage of. The transaction goes as follows:**

BEGIN TRANSACTION SUS\_ALERT

INSERT INTO KARMA ('11000', '12000', '10000', '10000', '0000');

IF error THEN GO TO UNDO; END IF;

IF Buyer\_Flag == 0 && Karma\_Balance >= 10000 THEN GO TO SUSPEND; END IF;

SUSPEND:

UPDATE Buyer\_Flag == 1;

IF error THEN GO TO UNDO; END IF;

COMMIT;

GO tO FINISH;

UNDO:

ROLLBACK;

FINISH:

END TRANSACTION;

**3a Team Member Contributions**

Rio was very helpful throughout the duration of the project. His main contribution is the SQL DB itself, and he did a lot of work with the create and insert SQL, as well as helping out with the sql queries. He also had a major role in the creation of the relational schema. He made the original ERD that we submitted for CP01, which was useful as a reference in our final ERD. He also created the Google Drive that we stored our work in, and created the format for the checkpoint documents.

Daniel helped by creating the final ERD after getting some additional feedback on CP01 that we should revert to an earlier design that was similar to one of his drafts. He played a major role with the additional entities of Wishlist and Store Categories. He worked with the relational algebra in detail. He wrote some of the SQL queries, and also worked on the insert queries. He led work on the normalization of the database, and also helped with indexing and views.

Avi helped by contributing to the ERD design. He did a lot of work with the relational schema, going through the process of creating it with Rio. He did a lot of work on the relational algebra, and also did a major portion of the SQL queries.

Tyler joined our group late, but still was helpful by working with us on the relational schema. He also did a lot of work with inserts for the SQL. He worked on SQL queries after the inserts were finished up. He also did the SQL transactions.

**3b Reflection of Projection Completion**

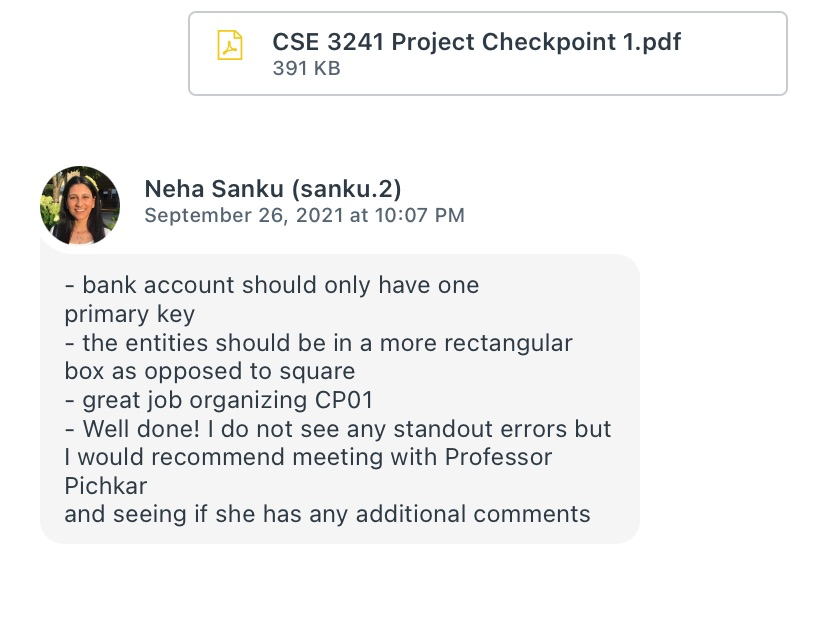
Our team aimed to meet either in person or virtually once a week most weeks. This was made difficult due to our schedules, but we were still able to have many meetings, including some at the library, particularly before checkpoints were due, which we found helpful. It was difficult to design a project of this size, particularly at the start of the project, where were just in the introductory stages of the class, and tasked with designing our own database.

For this project, we would recommend pushing the deadlines for the checkpoints further back into the semester, and having the homeworks represent the final project better, both in difficulty and in content. It was difficult to work on the checkpoints when we were still learning the material 1-2 weeks before the checkpoint is due. By ensuring that all the content needed for the checkpoints is covered at least a week, ideally two, before the checkpoint is due, better work could be done on the checkpoints, which would help with the overall project quality and help prevent major revisions from having to be made.

By making homeworks center around the homework, students would be better able to handle the requirements of the checkpoints. A greater focus on the design aspects of creating an EERD and Schema, as well as a lot more practice with SQL queries, would be appreciated. The homework was generally more conceptual, or did not line up with the difficulty of the project.

**3c Description of Feedback and Revisions**

**CP01**

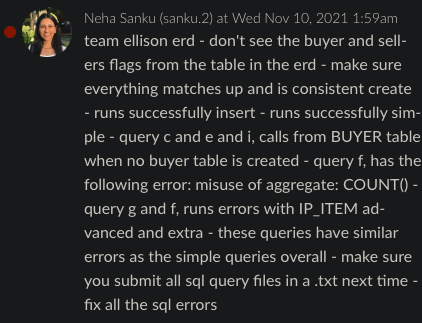
****

We ended up meeting with Dr. Pichkar and eventually creating a new version of the ERD. When we remade the ERD we redid Payment Methods, which includes the Bank Account entity, and we made sure to put the entities into more rectangular boxes.

**CP02**

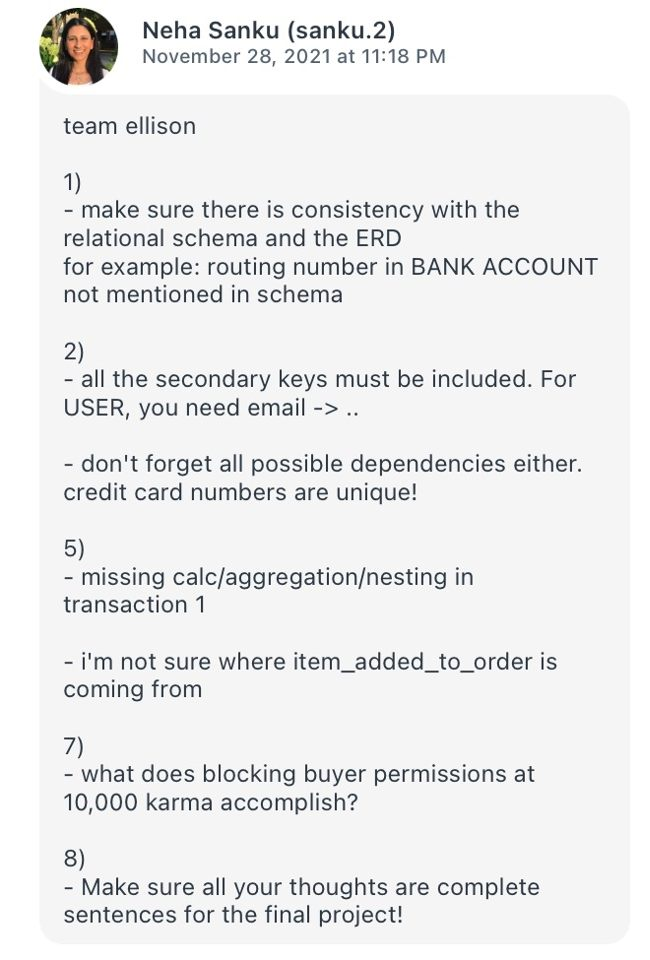
Need image

**CP03**

****

We addressed the missing buyer and seller flags, and worked to increase the stability and fix the errors of our SQL. We adjusted the queries mentioned to prevent misusing aggregate functions and to prevent errors with IP\_Item.

**CP04**

****

-We went through our ERD and relational schema again to ensure that everything was consistent.

-We redid our normalization to consider candidate keys and other possible dependencies.

-We changed transaction 1 to ensure that it used calc/aggregation/nesting. Item\_Added\_To\_Order is the name of our N:M relationship between IP\_Item and Item\_Order (our items and our orders).

-We clarified the purposes of our transactions.

**3d Project Checkpoints**

**CP01**

Team Members:

* 1. Rio Ellison.1616
  2. Daniel Mills.778
  3. Avi Popat.5
  4. Tyler Simpson.729

The team has planned to meet at least twice per week for approximately one to two hours soon after each lecture, with additional meetings as needed per mutual availability, either virtually or in-person at their discretion. In-person meetings may take place at an on-campus location agreed upon by the team while virtual meetings will be conducted via an online and web conferencing platform, i.e., Zoom.

Communication amongst team members between meetings will be conducted via GroupMe, a mobile group messaging app. A Google Drive folder granting editor permissions to all team members will be used to create and update all documents relevant to the semester-long team project.

Team members should not expect difficulties regarding different time zones or technology access. The team will initially address and attempt to resolve potential issues regarding the project should they arise. If the issue(s) cannot be resolved internally, any team member(s) may request assistance from the course instructor.

The following is a working list of entities with associated attributes to be modeled in the database, given the current requirements in the project overview:

Note: Solid bullet points denote Entities; hollow bullet points denote Attributes; boxed points denote an attribute of an attribute, i.e., parent attribute of boxed point(s) assumed to be a compound attribute.

* Buyer
  + Buyer ID
  + Email
  + Password
  + Name
    - First Name
    - Last Name
* Seller
  + Seller ID
  + Email
  + Password
  + Name
    - First Name
    - Last Name
* IP Item
  + Item ID
  + Title
  + Description
  + Price
  + File Type
  + Images (multi)
  + Keywords (multi)
* Cart
  + Cart ID
  + Subtotal (derived)
  + IP Item List (multi)
* Order
  + Order ID
  + Total Cost (derived)
  + IP Item List (multi)
* Transaction
  + Transaction ID
* Buyer Financial History
  + Record ID
* Purchase
  + Purchase ID
  + Purchase Date
  + IP Item List (multi)
* Seller Financial History
  + Record ID
  + Sales
  + Expenses
  + Profit (derived)
* Payment Method
  + Payment ID
  + Balance (derived)
* Bank Account
  + Account Number
  + Routing Number
  + Account Holder Name
* Credit Card
  + Card Holder Name
  + Card Number
  + CVV
* Karma
  + Karma Wallet ID
* Crypto
  + Crypto Wallet ID
  + Encryption Key
* Store
  + Store ID
  + Name
  + Description
  + Banner
  + Seller Bio
  + Seller Photo
  + URLs (mutli)
* Store Categories
  + Category IDs
  + Names
* ‘Gives Feedback To’ Relation (Buyer and Product)
  + Numerical Rating
  + Description
* ‘Gives Feedback To’ Relation (Buyer and Seller)
  + Numerical Rating
  + Description
* ‘Sent To’ Relation (Buyer and Order)
  + Delivery Email
* ‘Confirms’ Relation (Transaction and Buyer)
  + Confirmation ID

The following is a working list of relationships between entities based on the overview:

A seller creates a store.

A seller gives feedback to the buyer.

A seller lists IP items.

A seller confirms transactions.

Seller accesses seller financial account.

A buyer gives feedback about the IP item.

A buyer confirms transactions.

A buyer accesses the buyer's financial account.

A buyer views a cart.

A buyer selects IP items.

Transactions are recorded in the buyer's and seller’s financial accounts.

A store contains IP items.

An IP item is added to the cart.

The cart becomes an order.

An order is sent to the buyer.

Order prompts transactions.

Transactions require a payment method.

Two additional entities useful to the database may include store categories and buy/seller chat rooms.

For our two additional entities, we propose a Wishlist entity and a Store Categories entity.

The Wishlist entity would be beneficial, as it would allow users to save IP items that they would like to buy at a later date. This data could also be aggregated, and could be used as a measure of an item's popularity, the more wishlisted an item is, the more desired it could be presumed to be. This could be a way for users to find more IP items to purchase.

The Wishlist entity would have a ‘Manages’ relationship with the buyer. A buyer can only have 1 wishlist, and they must have a wishlist (even if the wishlist is empty). A wishlist is assigned to every buyer, so can be assigned to n buyers. It would also have a ‘Contains’ relationship with the IP item. A wishlist can contain n amount of IP items but does not have to contain any. An IP item can be in m wishlists but does not have to be in any.

The Wishlist entity would have a key attribute of 'Wishlist ID'. There would be multiple "Contains" relationship attributes with an IP item, including 'Ranking' and 'Date Added'. 'Ranking' would allow for the user to rank IP items on their wishlist in an order of their choosing. 'Date added' would record when the user added an IP item to their wishlist, so they could see how long they wished for the IP item.

The Store Categories entity would be beneficial, as it would allow for labeling stores by what sort of IP items they sell. This could allow for measuring which categories are most popular on the site. For example, if stores of a certain category turn out to be more popular, incentives could be made by the store to encourage sellers to create more of them. It would also allow buyers to more easily find IP items that are of interest to them, as they can browse categories that they like. Finally, if the stores are ordered per category, the website could push more popular stores to the top, which are more likely to make a sale or reward certain stores by placing them at the top of a category.

The Store Categories entity would have an ‘Assigned To’ attribute with Stores. A store can be in up to 3 categories but does not have to be in any. A store category can have n amount of stores but does not have to be in any.

The Store Categories entity would have a key attribute of 'Name'. It would not be possible to create two categories with the same name. The Store Categories entity will also have an attribute with its "Contained In" relationship between stores. This attribute would be 'Ranking'. Stores would be ranked based on their popularity, which could be measured by IP items purchased from the store, or perhaps the number of people who have an item from the store on their wishlist.

5.

A query for store categories could be a “category overview” page, which would list the number of stores and total number of IP items available for each category. This would allow buyers to easily see which categories are the most popular, and provide a convenient list for them to find categories that they are interested in. This would also allow sellers to see which categories have the most saturated market.

A query for a wishlist could be to create a “most-wished” item page, so users can see what items are most desired by buyers. This could allow buyers to find high-quality IP items that a lot of users want. This could also allow sellers to see which items are most popular with buyers.

Another query could allow sellers to see a list of the stores that they have created. This could include valuable information such as the number of IP items per store and the average rating of the IP items in the store

Another query could be a seller rating page. Sellers could be ranked based on the average rating. The stores that the seller manages could be listed alongside their entry. This would allow buyers to find high-quality sellers and easily access their stores.

6.

Given the entities and relationships outlined above, a new IP item will be added to the database by the seller, who accesses the database and creates a listing of the IP item and its relevant info, e.g., title, description, price, all contained within the store entity. This applies to all sellers adding some arbitrary number of IP items to their respective virtual stores.

Each IP listing created by a seller will include the option to include up to five images of the IP item, viewable to both buyers and sellers alike.

A Buyer may choose to complete a transaction with any combination of available payment methods, i.e., karma, cryptocurrency, dollars. Various payment IDs, i.e., attribute data, will be included depending on the option(s) chosen for payment by the Buyer.

A Buyer may purchase multiple IP items from multiple Sellers at the same time as part of one order. Once a Buyer has IP items from a single or multiple sellers added to the Cart, the order prompts individual transaction requests between the Buyer and each Seller, with each item in the Order sent to the buyer once they and the respective Buyer both confirm on the transaction.

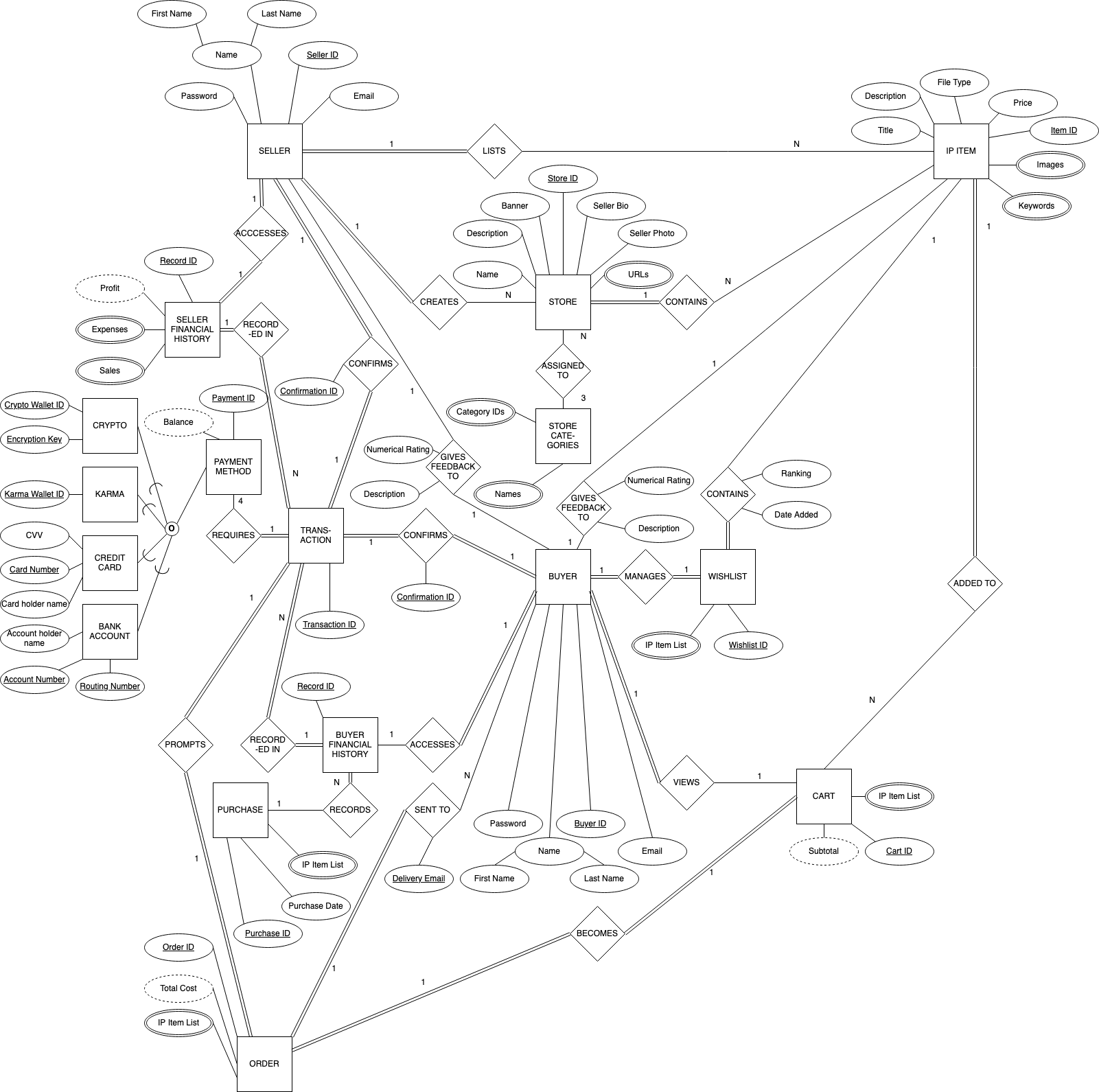
A Buyer may only leave feedback for an item they have purchased and received from a particular Seller/Store. Additional attributes may be included to check Buyer status allowing them to post feedback.

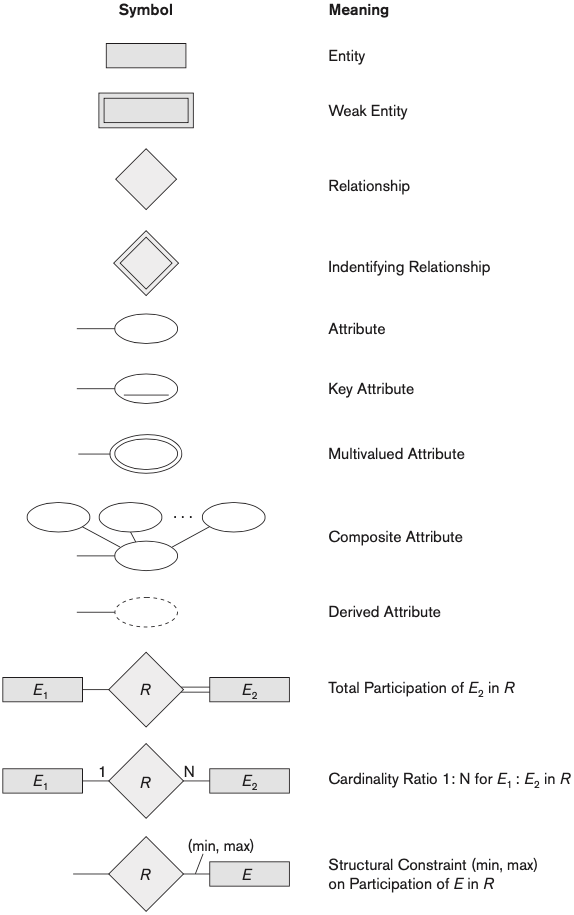
8.

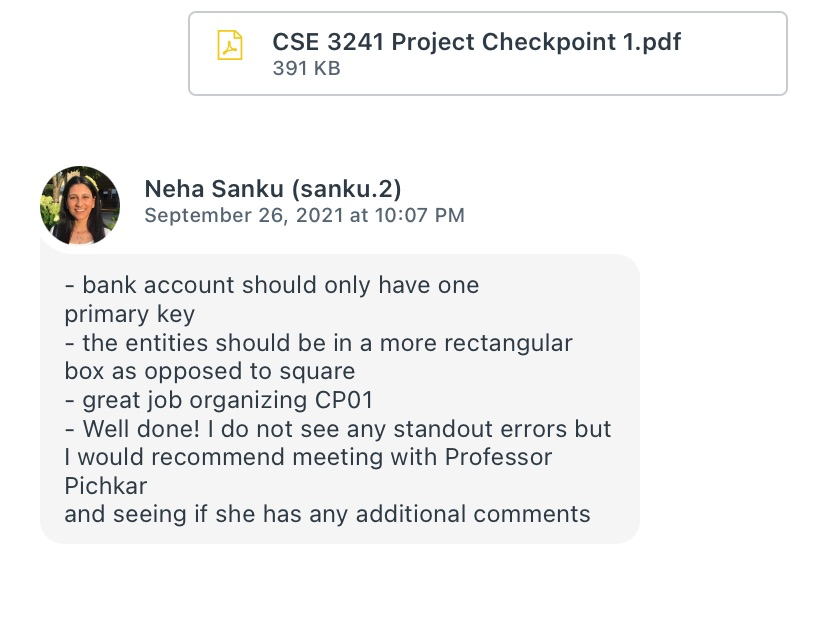
A store category could be added as an update operation. Creating a new category would help to more closely define the kinds of categories. Stores would have to be updated to be listed as part of this new category.

A wishlist could be cleared as an update attribute. If a user wants to start over with a fresh wishlist, they could clear all items from their list. This would only involve removing the IP items from the user’s particular list. The user would presumably add more IP items to their list after doing this operation.

An IP item could have its attributes changed. This would involve changing attributes such as the title, description, and could even extend to changing which stores the item is listed in, which would affect which IP items a store lists. An application of this could be putting an item on sale.

9.



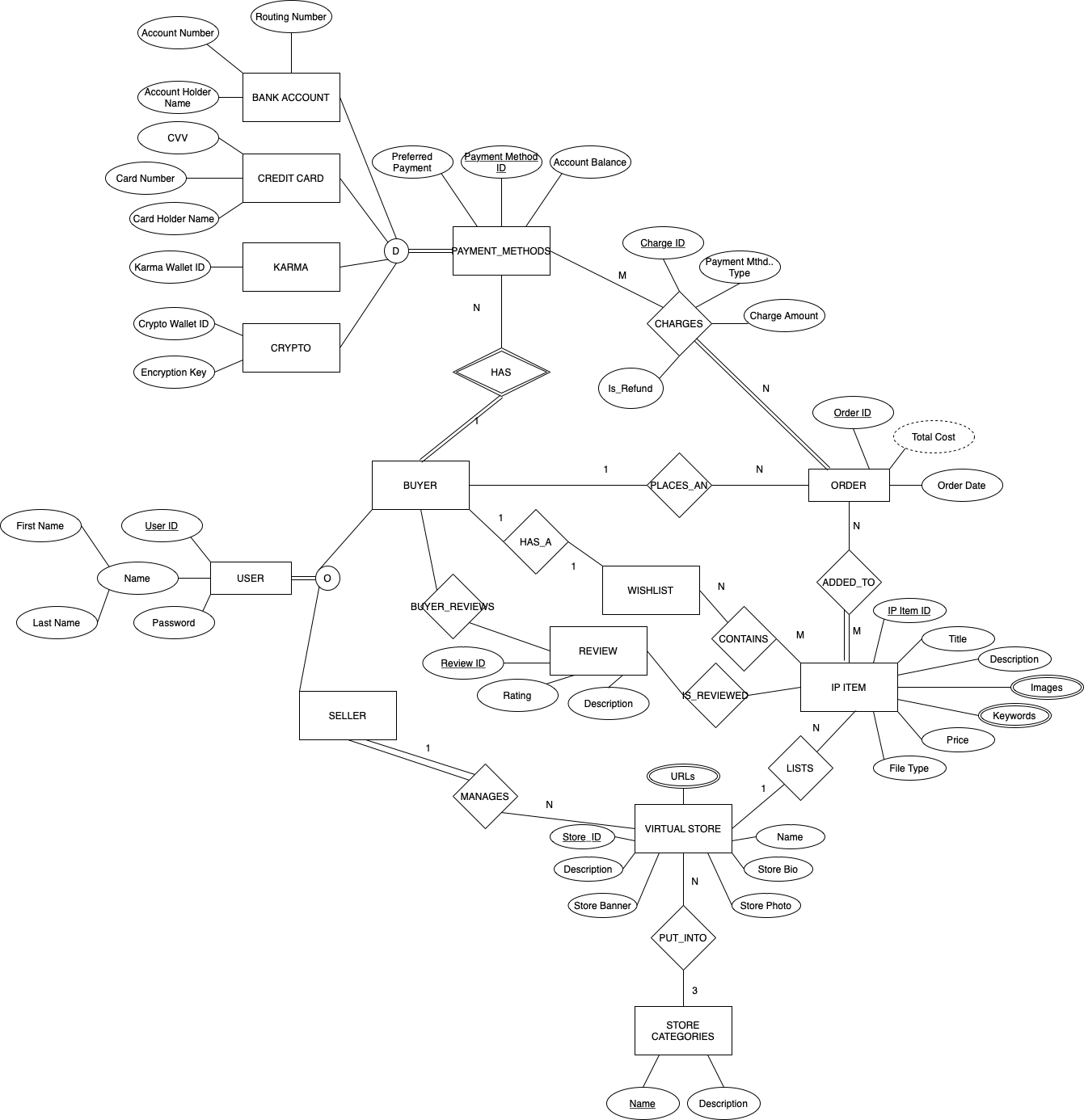
****

**CP02**

1.

Below is a current version of the team’s ER Model. The following revisions include:

* EditED square entities to rectangular
* Pending: Bank account entity to only include one key attribute



2.

**USER** (User ID, First Name, Last Name, Password, Buyer\_flag, Seller\_flag, Email)

**WISHLIST** (Wishlist ID**,** User ID (FK))

User ID is a foreign key originating from **USER**.

**ITEMS\_IN\_WISHLIST** (Wishlist ID (FK), IP Item ID (FK))

Wishlist ID and IP Item ID are foreign keys originating from **WISHLIST** and **IP ITEM**, respectively.

**REVIEW** (Review ID, Rating, Description, IP Item ID (FK), User ID (FK))

IP Item ID and User ID are foreign keys originating from **IP ITEM** and **USER**, respectively.

**BANK ACCOUNT** (Payment Method ID (FK), Account Holder Name, Account Number, Charge ID (FK), User ID (FK))

User ID is a foreign key originating from **USER** and Payment Method ID and Charge ID are foriegn keys originating from **PAYMENT\_METHODS**.

**CREDIT CARD** (Payment Method ID (FK), CVV, Card Number, Cardholder Name, Charge ID (FK), User ID (FK))

User ID is a foreign key originating from **USER** and Payment Method ID and Charge ID are foriegn keys originating from **PAYMENT\_METHODS**.

**KARMA** (Payment Method ID (FK), Karma Wallet ID, Karma Balance, Charge ID (FK), User ID (FK))

User ID is a foreign key originating from **USER** and Payment Method ID and Charge ID are foriegn keys originating from **PAYMENT\_METHODS**.

**CRYPTO** (Payment Method ID (FK), Crypto Wallet ID, Charge ID (FK), User ID (FK))

User ID is a foreign key originating from **USER** and Payment Method ID and Charge ID are foriegn keys originating from **PAYMENT\_METHODS**.

**ITEM\_ORDER** (Order ID, Order Date, Is Refund, Price (FK), User ID (FK))

User ID is a foreign key originating from **USER**. Price is a foreign key originating from **IP\_ITEM**.

**CHARGES** (Order ID (FK), Payment Method Type (FK), Charge Amount)

Order ID and Payment Method Type are both foreign keys originating from **ITEM\_ORDER** and **PAYMENT\_METHODS**, respectively.

**ITEM\_ADDED\_TO\_ORDER** (Order ID (FK), IP Item ID (FK))

Order ID and IP Item ID are both foreign keys originating from **ORDER** and **IP ITEM**, respectively.

**IP ITEM** (IP Item ID, Store ID (FK), Title, IP-Description, Price, File Type)

Store ID is a foriegn key originating from **VIRTUAL STORE**.

**IP ITEM IMAGES** (Image ID, IP Item ID (FK), Image description, Image name, File type, Upload date)

IP Item ID is a foreign key originating from **IP ITEM**.

**IP ITEM KEYWORDS** (Keyword ID, IP Item ID (FK), Keyword Name, Upload Date)

IP Item ID is a foriegn key originating from **IP ITEM**.

**VIRTUAL STORE** (Store ID, User ID (FK), Store Description, Store Banner, Name, Store Bio, Store Photo, Store Category ID)

User ID is a foreign key originating from **USER**.

Store\_Category is a foreign key originating from Category Name in **STORE CATEGORIES.**

**VIRTUAL STORE URLS** (URL ID, URL Name, URL Link, Store ID (FK), Upload Date)

Store ID is a foreign key originating from **VIRTUAL STORE**.

**IP ITEM REVIEWED** (IP Item ID, Review ID)

IP Item ID is a foreign key originating from **IP ITEM**

Review ID is a foregin key originating from **REVIEW**

**ORDER REVIEWED** (Order ID, Review ID)

Order ID is a foreign key originating from **ORDER**

Review ID is a foreign key originating from **REVIEW**

**STORE CATEGORIES** (Store Category ID, Category Name, Description)

3.

The following are a list of relational algebra expressions for queries from the relational schema

1. Create a list of IP items and the stores selling those.

IP\_ITEMS\_AND\_STORES ← VIRTUAL\_STORE \* IP\_ITEM

*π*(Title, Name)(IP\_ITEMS\_AND\_STORES)

1. Find the titles of all IP items that cost less than $10.

*πtitle (σ*(Price < 10)(IP\_Item))

1. Generate a list of IP item titles and dates of purchases made by a given buyer (you choose how to designate a buyer).

SOME\_USER ← *σ*(User ID = 0x64bb757d)(Order)

SELECTED\_USER ← SOME\_USER \* ITEM\_ADDED\_TO\_ORDER

SELECTED\_USER\_ORDER\_IP ← SELECTED\_USER \* IP\_ITEM

*π*(Title, Order Date)SELECTED\_USER\_ORDER\_IP

1. List all the buyers who purchased an IP item from a given store (you choose how to designate a store) and the names of the IP items they purchased.

Relations needed: VIRTUAL\_STORE, IP\_ITEM

SOME\_STORE ← *σ*(Store ID = 19384768)(IP\_ITEM)

SELECTED\_STORE ← SOME\_STORE \* ITEM\_ADDED\_TO\_ORDER

SELECTED\_STORE\_ITEM\_IP ← SELECTED\_STORE \* ORDER

FINAL\_IP\_ITEM\_BUYER\_LIST ← SELECTED\_STORE\_ITEM\_UP \* USER

*π*(First Name, Last Name, Title)FINAL\_IP\_ITEM\_BUYER\_LIST

1. Find the buyer who has purchased the most IP items and the total number of IP items they have purchased.

SOME\_ORDER ← ITEM\_ADDED\_TO\_ORDER \* ORDER

SOME\_USER\_ID ←User IDFCOUNT IP\_ITEM(SOME\_ORDER)

MOST\_ITEM\_BOUGHT(User ID, Total Items) ← User IDFMAX IP Item ID(SOME\_USER\_ID)

1. Create a list of stores who currently offer 5 or less IP items for sale.

STORES\_BY\_SOLD\_ITEMS(Store ID, Item Count) ← Store IDFCOUNT IP Item ID(IP\_ITEM)

*σ*Item Count <= 5(STORES\_BY\_SOLD\_ITEMS)

1. 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.

ITEM\_COUNT(IP Item ID, Total Sales, Quantity) ← Item IDFSUM Price,COUNT IP Item ID(ITEM\_ADDED\_TO\_ORDER)

MOST\_SOLD\_ITEM(Item ID, No. of Items Sold) ← IP Item IDFMAX Count(ITEM\_COUNT)

IP\_ITEM\_SALES ← ITEM\_COUNT \* IP\_ITEM

INTERMEDIATE\_ITEM\_LIST ← MOST\_SOLD\_ITEM \* IP\_ITEM\_SALES

*π*Item ID, Total Sales, No. of Items Sold, Store ID(INTERMEDIATE\_ITEM\_LIST)

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

**ACCEPTED PAYMENT TYPES**

| Payment Type | Times Used | Total Charge |
| --- | --- | --- |

CHARGES\_BY\_PAYMENT\_TYPE(Payment Types, Num Payment Types) ← Payment TypesFCOUNT Charge ID(Charges)

CHARGE\_SUMS\_BY\_PAYMENT\_TYPE(Payment Types, Total Charge) ← Payment TypesFSUM Charge Amount(Charges)

ACCEPTED\_PAYMENT\_TYPES ← *π*Payment Types, Num Payment Types, Total Charge(CHARGE\_SUMS\_BY\_PAYMENT\_TYPE \* CHARGES\_BY\_PAYMENT\_TYPE)

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

USER\_KARMA\_BAL(Max Account Balance) ←MAX Account Balance(KARMA)

INTERMEDIATE\_KARMA\_BAL ← KARMA ⋈MAX Account Balance = Account Balance USER\_KARMA\_BAL

INTERMEDIATE\_KARMA\_BAL2 ← USER \* INTERMEDIATE\_KARMA\_BAL

*π*First Name, Last Name, Email Account Balance(INTERMEDIATE\_KARMA\_BAL2)

4.

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

How many stores are in each category: Category NameFCOUNT Store ID(STORE\_CATEGORY)

Find how many wishlists each IP Item is on: IP Item IDFCOUNT Wishlist ID(ITEMS\_IN\_WISHLIST)

Average cost of item on each store: IP Item IDFAVERAGE Price(VIRTUAL\_STORE)

**NEED IMAGE**

**CP03**

/\*

3a

Create a list of IP items and the stores selling those.

\*/

SELECT IP.Title, VS.Name

FROM IP\_ITEM AS IP, VIRTUAL\_STORE AS VS;

/\*

3b

Find the titles of all IP Items that cost less than $10.

\*/

SELECT IP.Title

FROM IP\_ITEM AS IP

WHERE IP.Price < 10;

/\*

3c

Generate a list of IP item titles and dates of purchase made by a given buyer (you choose how to designate a buyer).

\*/

SELECT IP.Title, IO.Order\_Date

FROM BUYER AS B, ITEM\_ORDER AS IO, ITEM\_ADDED\_TO\_ORDER AS IATO, IP\_ITEM AS IP

WHERE B.User\_ID = ‘727’

AND B.User\_ID = IO.User\_ID

AND IO.Order\_ID = IATO.Order\_ID

AND IATO.IP\_Item\_ID = IP.IP\_Item\_ID;

/\*

3d

List all the buyers who purchased an IP Item from a given store (you choose how to designate a store) and the names of the IP Items they purchased.

\*/

SELECT B.First\_Name, B.Last\_Name

FROM BUYER AS B, ITEM\_ORDER AS IO, ITEM\_ADDED\_TO\_ORDER AS IATO, IP\_ITEM AS IP, VIRTUAL\_STORE AS VS

WHERE VS.Store\_ID = ‘4545’

AND VS.Store\_ID = IP.Store\_ID

AND IP.IP\_Item\_ID = IATO.IP\_Item\_ID

AND IATO.Order\_ID = IO.Order\_ID

AND IO.User\_ID = B.User\_ID;

/\*

3e

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

\*/

SELECT B.First\_Name, B.Last\_Name, COUNT(IP\_Item.ID)

FROM BUYER AS B, ITEM\_ORDER AS IO, ITEM\_ADDED\_TO\_ORDER AS IATO

WHERE B.User\_ID = IO.User\_ID

AND IO.Order\_ID = IATO.Order\_ID

AND MAX(IP\_Item.ID)

GROUP BY B.User\_ID;

/\*

3f

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

\*/

SELECT VS.Name

FROM VIRTUAL\_STORE AS VS, IP\_ITEM AS IP

WHERE VS.Store\_ID = IP.Store\_ID

AND COUNT(VS.Store\_ID) < 5

GROUP BY VS.Store\_ID;

/\*

3g

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.

\*/

SELECT IP.Title, revenue, units\_sold, Store\_ID FROM

(SELECT IP.Title, COUNT(Order\_ID)\*IP.Price AS revenue, COUNT(Order\_ID) as units\_sold, Store\_ID FROM IP\_ITEM AS IP, ITEM\_ADDED\_TO\_ORDER AS IATO

WHERE IP.IP\_Item\_ID = IATO.IP\_Item\_ID

GROUP BY IP.IP\_Item\_ID

ORDER BY revenue DESC);

/\*

3h

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.

\*/

SELECT C.Payment\_Method\_Type, COUNT(C.Payment\_Method\_Type), SUM(IP.Price)

FROM CHARGES AS C, ITEM\_ORDER AS IO, ITEM\_ADDED\_TO\_ORDER AS IATO

WHERE C.Order\_ID = IO.Order\_ID

AND IO.Order\_ID = IATO.Order\_ID

GROUP BY C.Payment\_Method\_Type;

/\*

3i

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

\*/

SELECT B.First\_Name, B.Last\_Name, B.Email

FROM KARMA AS K, BUYER AS B

WHERE MAX(Karma\_Balance);

/\*

4a

How many stores are in each category

\*/

SELECT SC.Category\_Name, COUNT(VS.Store\_ID)

FROM STORE\_CATEGORIES AS SC, VIRTUAL\_STORE AS VS

WHERE SC.Store\_Category\_ID = VS.Store\_Category\_ID

GROUP BY Store\_ID;

/\*

4b

Find how many wishlists each IP Item is on

\*/

SELECT IIW.IP\_Item\_Name, COUNT(IIW.IP\_Item\_ID)

FROM ITEMS\_IN\_WISHLIST AS IIW

GROUP BY IIW.IP\_Item\_Name;

/\*

4c

Average cost of item on each store

\*/

SELECT VS.Name, AVG(IP.Price)

FROM VIRTUAL\_STORE AS VS, IP\_ITEM AS IP

WHERE IP.Store\_ID = VS.Store\_ID

GROUP BY VS.Store\_ID;

/\*

5a

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

\*/

SELECT B.First\_Name, B.Last\_Name, SUM(IP.Price)

FROM BUYER AS B, ITEM\_ORDER AS IO, ITEM\_ADDED\_TO\_ORDER AS IATO, IP\_ITEM AS IP

WHERE B.User\_ID = IO.User\_ID

AND IO.Order\_ID = IATO.Order\_ID

AND IATO.IP\_Item\_ID = IP.IP\_Item\_ID

GROUP BY B.User\_ID;

/\*

5b

Provide a list of buyer names and e-mail addresses for buyers who have spent more than the average buyer.

\*/

SELECT B.First\_Name, B.Last\_Name, B.Email

FROM BUYER AS B, ITEM\_ORDER AS IO, ITEM\_ADDED\_TO\_ORDER AS IATO, IP\_ITEM AS IP

WHERE B.User\_ID = IO.User\_ID

AND IO.Order\_ID = IATO.Order\_ID

AND IATO.IP\_Item\_ID = IP.IP\_Item\_ID

GROUP BY B.User\_ID

HAVING SUM(IP.Price) >

(SELECT AVG(total\_price)

FROM (SELECT SUM(IP.Price) AS total\_price FROM USER AS B, ITEM\_ORDER AS IO, ITEM\_ADDED\_TO\_ORDER AS IATO, IP\_ITEM AS IP

WHERE B.User\_ID = IO.User\_ID

AND IO.Order\_ID = IATO.Order\_ID

AND IATO.IP\_Item\_ID = IP.IP\_Item\_ID

GROUP BY B.User\_ID));

/\*

5c

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 IP.Title, COUNT(IP.IP\_Item\_ID)

FROM IP\_ITEM AS IP, ITEM\_ADDED\_TO\_ORDER AS IATO

WHERE IP.IP\_Item\_ID = IATO.IP\_Item\_ID

GROUP BY IP.IP\_Item\_ID

ORDER BY COUNT(IP.IP\_Item\_ID) DESC;

/\*

5d

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 IP.Title, COUNT(Order ID)\*IP.Price AS revenue

FROM IP\_ITEM AS IP, ITEM\_ADDED\_TO\_ORDER AS IATO

WHERE IP.IP\_Item\_ID = IATO.IP\_Item\_ID

GROUP BY IP.IP\_Item\_ID

ORDER BY revenue DESC;

/\*

5e

Find the most popular seller (i.e. the one who has sold the most IP Items)

\*/

SELECT S.First\_Name, S.Last\_Name, MAX(items\_sold)

FROM (SELECT S.First\_Name, S.Last\_Name, COUNT(IP\_Item\_ID) AS items\_sold FROM ITEM\_ADDED\_TO\_ORDER AS IATO, IP\_ITEM AS IP, VIRTUAL\_STORE AS VS, USER AS S

WHERE IATO.IP\_Item\_ID = IP.IP\_Item\_ID

AND IP.Store\_ID = VS.Store\_ID

AND VS.User\_ID = S.User\_ID

GROUP BY S.User\_ID);

/\*

5f

Find the most profitable seller (i.e. the one who has brought in the most money)

\*/

SELECT S.First\_Name, S.Last\_Name, MAX(profit)

FROM (SELECT S.First\_Name, S.Last\_Name, SUM(IP.Price) AS profit FROM ITEM\_ADDED\_TO\_ORDER AS IATO, IP\_ITEM AS IP, VIRTUAL\_STORE AS VS, USER AS S

WHERE IATO.IP\_Item\_ID = IP.IP\_Item\_ID

AND IP.Store\_ID = VS.Store\_ID

AND VS.User\_ID = S.User\_ID

GROUP BY S.User\_ID);

/\*

5g

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

\*/

SELECT B.First\_Name, B.Last\_Name

FROM User as B

WHERE Order\_ID IN

SELECT S.User\_ID, MAX(profit)

FROM (SELECT S.First\_Name, S.Last\_Name, SUM(IP.Price) AS profit FROM ITEM\_ADDED\_TO\_ORDER AS IATO, IP\_ITEM AS IP, VIRTUAL\_STORE AS VS, USER AS S

WHERE IATO.IP\_Item\_ID = IP.IP\_Item\_ID

AND IP.Store\_ID = VS.Store\_ID

AND VS.User\_ID = S.User\_ID

GROUP BY S.User\_ID);

/\*

5h

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

\*/

/\*

5i

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

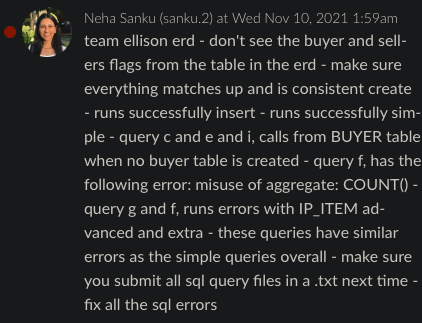
\*/

SELECT VS.Name, COUNT(IP.IP\_Item\_ID), MAX(IP.Price), MIN(IP.Price), AVG(IP.Price)

FROM VIRTUAL\_STORE AS VS, IP\_ITEM AS IP

WHERE VS.Store\_ID = ‘4545’

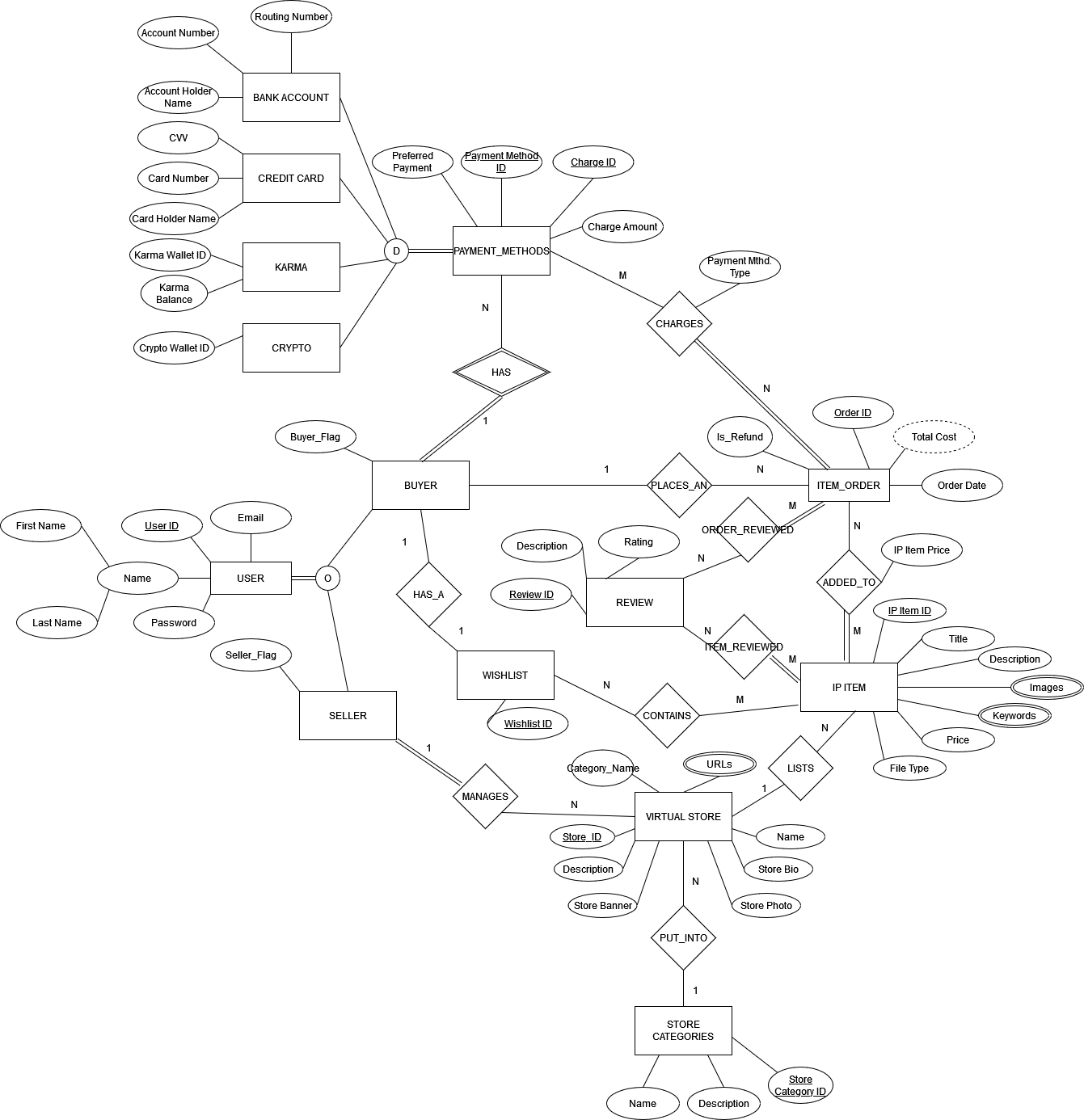
AND VS.Store\_ID = IP.Store\_ID

****

**CP04**

1.

Provide a current version of your ER Diagram and Relational Model as per Project Checkpoint 03. If you were instructed to change the model for Project Checkpoint 03, make sure you use the revised versions of your models.



**USER** (User\_ID, First\_Name, Last\_Name, Password, Buyer\_Flag, Seller\_Flag, Email)

**WISHLIST** (Wishlist\_ID**,** User\_ID (FK))

User ID is a foreign key originating from **USER**.

**ITEMS\_IN\_WISHLIST** (Wishlist\_ID\_(FK), IP\_Item\_ID (FK))

Wishlist ID and IP Item ID are foreign keys originating from **WISHLIST** and **IP ITEM**, respectively.

**REVIEW** (Review\_ID, Rating, Description, IP\_Item\_ID (FK), User\_ID (FK))

IP Item ID and User ID are foreign keys originating from **IP ITEM** and **USER**, respectively.

**BANK ACCOUNT** (Payment Method ID (FK), Account Holder Name, Account Number, Charge ID (FK), User ID (FK))

User ID is a foreign key originating from **USER** and Payment Method ID and Charge ID are foriegn keys originating from **PAYMENT\_METHODS**.

**CREDIT CARD** (Payment Method ID (FK), CVV, Card Number, Cardholder Name, Charge ID (FK), User ID (FK))

User ID is a foreign key originating from **USER** and Payment Method ID and Charge ID are foriegn keys originating from **PAYMENT\_METHODS**.

**KARMA** (Payment Method ID (FK), Karma Wallet ID, Karma Balance, Charge ID (FK), User ID (FK))

User ID is a foreign key originating from **USER** and Payment Method ID and Charge ID are foriegn keys originating from **PAYMENT\_METHODS**.

**CRYPTO** (Payment Method ID (FK), Crypto Wallet ID, Charge ID (FK), User ID (FK))

User ID is a foreign key originating from **USER** and Payment Method ID and Charge ID are foriegn keys originating from **PAYMENT\_METHODS**.

**ITEM\_ORDER** (Order ID, Order Date, Is Refund, Price (FK), User ID (FK))

User ID is a foreign key originating from **USER**. Price is a foreign key originating from **IP\_ITEM**.

**CHARGES** (Order ID (FK), Payment Method ID (FK), Charge Amount)

Order ID and Payment Method Type are both foreign keys originating from **ITEM\_ORDER** and **PAYMENT\_METHODS**, respectively.

**ITEM\_ADDED\_TO\_ORDER** (Order ID (FK), IP Item ID (FK))

Order ID and IP Item ID are both foreign keys originating from **ORDER** and **IP ITEM**, respectively.

**IP ITEM** (IP\_Item\_ID, Store\_ID (FK), Title, IP\_Description, Price, File\_Type)

Store ID is a foreign key originating from **VIRTUAL STORE**.

**IP ITEM IMAGES** (Image\_ID, IP\_Item\_ID (FK), Image\_Description, Image\_Name, File\_Type, Upload\_Date)

IP Item ID is a foreign key originating from **IP ITEM**.

**IP ITEM KEYWORDS** (Keyword\_ID, IP\_Item\_ID (FK), Keyword\_Name, Upload\_Date)

IP Item ID is a foriegn key originating from **IP ITEM**.

**VIRTUAL STORE** (Store ID, User\_ID (FK), Store\_Description, Store\_Banner, Name, Seller\_Photo, Store\_Category ID (FK))

User ID is a foreign key originating from **USER**.

Store\_Category is a foreign key originating from Category Name in **STORE CATEGORIES.**

**VIRTUAL STORE URLS** (URL\_ID, URL\_Name, URL\_Link, Store\_ID (FK), Upload\_Date)

Store ID is a foreign key originating from **VIRTUAL STORE**.

**IP ITEM REVIEWED** (IP Item ID, Review ID)

IP Item ID is a foreign key originating from **IP ITEM**

Review ID is a foregin key originating from **REVIEW**

**ORDER REVIEWED** (Order ID, Review ID)

Order ID is a foreign key originating from **ORDER**

Review ID is a foreign key originating from **REVIEW**

**STORE CATEGORIES** (Store\_Category ID, Category\_Name, Description, Store\_ID (FK))

Store ID is a foreign key originating from **VIRTUAL STORE**.

2.

Check that each relation in your schema is in 1NF and if they are not, bring them to 1NF. For each relation schema (table) in your model, indicate the functional dependencies. Make sure to consider all the possible dependencies in each relation and not just the ones from your primary keys.

**USER**

{User ID} -> {First\_Name, Last\_Name, Password, Buyer\_flag, Seller\_flag, Email}

**WISHLIST**

{Wishlist\_ID} -> {User ID}

**ITEMS\_IN\_WISHLIST**

{Wishlist\_ID, IP\_Item\_ID}

**REVIEW**

{Review\_ID} -> {Rating, Description, IP Item ID, User ID}

**BANK ACCOUNT**

{Payment\_Method\_ID} -> {Account Holder Name, Account Number, Charge ID, User ID}

**CREDIT CARD**

{Payment\_Method\_ID} -> {CVV, Card Number, Cardholder Name, Charge ID, User ID}

**KARMA**

{Payment\_Method\_ID} -> {Karma wallet ID, Karma Balance, Charge ID, User ID}

**CRYPTO**

{Payment\_Method\_ID} -> {Crypto Wallet ID, Charge ID, User ID}

**ITEM\_ORDER**

{Order\_ID} -> {Order\_Date, Is\_Refund}

**CHARGES**

{Order\_ID, Payment\_Method\_ID} -> {Charges}

**ITEM\_ADDED\_TO\_ORDER**

{Order\_ID, IP\_Item\_ID}

**IP ITEM**

{IP\_Item\_ID} -> {Store ID, Title, IP Description, Price, File Type}

**IP ITEM IMAGES**

{Image ID} -> {IP Item ID, Image Description, Image Name, File Type, Upload Date}

**IP ITEM KEYWORDS**

{Keyword ID} -> {IP Item ID, Keyword Name, Upload Date}

**VIRTUAL STORE**

{Store ID} -> {User ID, Store Description, Store Baner, Name, Store Bio, Store Photo, Store Category ID}

**VIRTUAL STORE URLS**

{URL ID} -> {URL Name, URL Link, Store ID, Upload Date}

**IP ITEM REVIEWED**

{IP Item ID, Review ID}

**ORDER REVIEWED**

{Order ID, Review ID}

**STORE CATEGORIES**

{Store Category ID} -> {Category Name, Description}

3.

For each relation schema in your model, determine the highest normal form of the relation. If the relation is not in 3NF, rewrite your relation schema so that it is in at least 3NF.

All relational schemas below are in 4th Normal Form.

**USER**

**WISHLIST**

**ITEMS\_IN\_WISHLIST**

**REVIEW**

**BANK ACCOUNT**

**CREDIT CARD**

**KARMA**

**CRYPTO**

**ITEM\_ORDER**

**CHARGES**

**ITEM\_ADDED\_TO\_ORDER**

**IP ITEM**

**IP ITEM IMAGES**

**IP ITEM KEYWORDS**

**VIRTUAL STORE**

**VIRTUAL STORE URLS**

**IP ITEM REVIEWED**

**ORDER REVIEWED**

**STORE CATEGORIES**

4.

For each relation schema in your model that is in 3NF but not in BCNF, either rewrite the relation schema to BCNF or provide a short justification for why this relation should be an exception to the rule of putting relations into BCNF.

All relational schemas below are in BCNF already.

**USER**

**WISHLIST**

**ITEMS\_IN\_WISHLIST**

**REVIEW**

**BANK ACCOUNT**

**CREDIT CARD**

**KARMA**

**CRYPTO**

**ITEM\_ORDER**

**CHARGES**

**ITEM\_ADDED\_TO\_ORDER**

**IP ITEM**

**IP ITEM IMAGES**

**IP ITEM KEYWORDS**

**VIRTUAL STORE**

**VIRTUAL STORE URLS**

**IP ITEM REVIEWED**

**ORDER REVIEWED**

**STORE CATEGORIES**

5.

For your database, propose at least two interesting views that can be built from your relations. These views must involve joining at least two tables together and must include calculations/aggregation/and/nesting. Provide SQL code for constructing your views along with the English language description of these views and what they do.

IDEA: IP Item alongside number of products sold, actively updating dashboard for sales

For our first view, we are going to join the IP\_ITEM and ITEM\_ADDED\_TO\_ORDER tables to create a dashboard for each item, and the number of times they have been purchased.

CREATE VIEW IPItemsSold

AS

SELECT IP.Title, IATO.Quantity

FROM IP\_ITEM as IP, ITEM\_ADDED\_TO\_ORDER as IATO

WHERE IP.IP\_Item\_ID = IATO.IP\_Item\_ID;

IDEA: Calculate the total costs of a user’s wishlist

For our second view, we join the USER, WISHLIST, ITEMS\_IN\_WISHLIST, and IP\_ITEM tables, this will allow us to see the sum of each product that a user has on their wishlist.

CREATE VIEW WishlistSum

AS

SELECT B.First\_Name, B.Last\_Name, SUM(IP.Price))

FROM USER as B, WISHLIST as W, ITEMS\_IN\_WISHLIST as IIW, IP\_ITEM as IP

WHERE B.User\_ID = W.User\_ID

AND W.Wishlist\_ID = IIW.Wishlist\_ID

AND IIW.IP\_Item\_ID = IP.IP\_Item\_ID

GROUP BY B.User\_ID;

6.

Description of two indexes that you want to implement in your DB. Explain their purpose and what you want to achieve by implementing them. Explain what type of indexing would be most appropriate for each one of them (Clustering, Hash, or B-tree) and why. To properly answer this question, look at your queries to identify the best candidates for indexing. Provide valid SQL code for each index.

IDEA: Price of IP Item, would do many range tests so should use a B-tree. Good because we often use the price in calculating profit

CREATE UNIQUE INDEX IP\_ITEM\_PRICE

ON IP\_ITEM (Price ASC);

IDEA: Virtual Store Name, would do equality tests so should use Hash. Good because we often need to list the name of the store in regards to the products it holds, or sale data from the store

CREATE UNIQUE INDEX STORE\_NAME

ON VIRTUAL\_STORE (Name ASC);

7.

Two sample transactions that you want to establish in your DB. Clearly document their purpose and function. Explain why it is crucial to execute each transaction you have created as one unit of processing. Each transaction should include read and write operations on at least two tables, with appropriate error and constraint checks and responses. Provide valid SQL code for each transaction.

IDEA: When a buyer completes the payment of an order, the transaction must go through both the buyer and seller's accounts. Meaning, if the payment fails on one end the whole transaction is canceled. We want to make sure the seller is receiving their payment when they sell an item.

BEGIN TRANSACTION COMPLETE\_PURCHASE

INSERT INTO KARMA ('11000', '12000', '1000', '10000', '0000');

IF error THEN GO TO UNDO; END IF;

UPDATE Payment\_Method\_ID SET Karma\_Balance = Karma\_Balance + 1000

WHERE Payment\_Method\_ID == 11000;

IF error THEN GO TO UNDO; END IF;

COMMIT;

GO TO FINISH;

UNDO:

ROLLBACK;

FINISH:

END TRANSACTION;

IDEA: When a User Karma balance is at least 10,000 Karma, their buyer permissions are restricted, i.e., unable to perform buyer actions.

BEGIN TRANSACTION SUS\_ALERT

INSERT INTO KARMA ('11000', '12000', '10000', '10000', '0000');

IF error THEN GO TO UNDO; END IF;

IF Buyer\_Flag == 0 && Karma\_Balance >= 10000 THEN GO TO SUSPEND; END IF;

SUSPEND:

UPDATE Buyer\_Flag == 1;

IF error THEN GO TO UNDO; END IF;

COMMIT;

GO tO FINISH;

UNDO:

ROLLBACK;

FINISH:

END TRANSACTION;

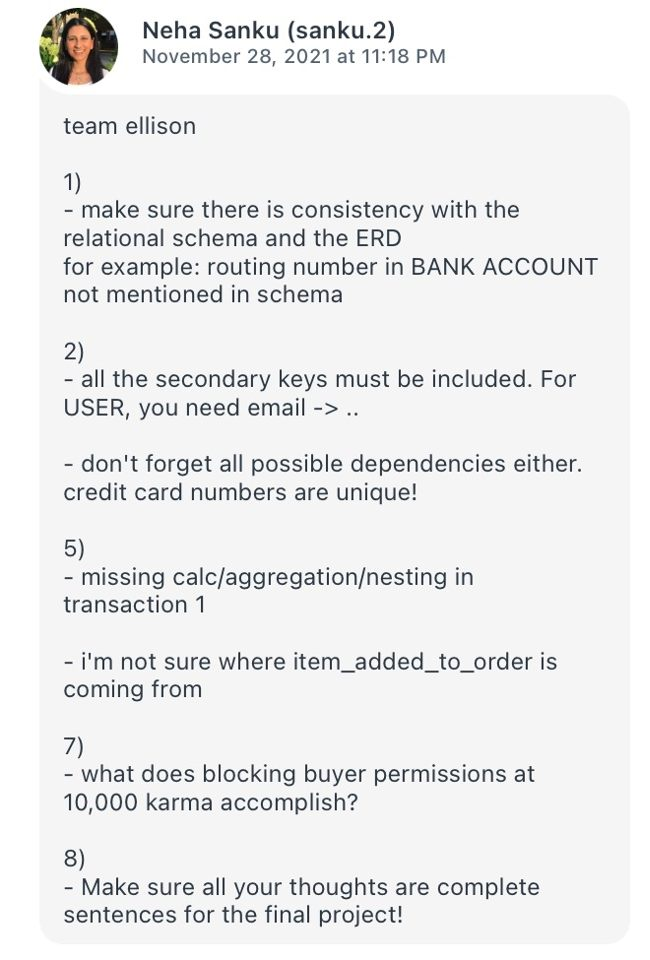
8.

Think about your DB design and the buyer/seller financial portal needs and possible inquiries and reports they may have. If this was not a classical relational DB, what data would you want to pre-aggregate/calculate and store in order to speed up information retrieval and meet user’s information needs? Provide at least 3 separate examples. Do not make any changes to your DB design or SQL code based on this question.

For the financial portal, we would want to pre-aggregate all of a buyer’s purchases and all of a seller’s sales within

We would also want to precalculate the revenue for each seller, since that value could be used in lots of other calculations, such as profit.

A user may wish to view their change of spending habits from some time in the past compared to now, e.g., % increase/decrease in Karma spending between this month and last month. An alternative system would pre-aggregate relevant primary attributes, i.e., IDs, relevant to these charges, e.g., User ID, Charge ID, Payment Method ID, Charge Amount, and calculate the percentage increase/decrease based on monthly intervals.

****