Database System for Alibaba E-commerce Public Dataset

Jiumu Zhu
New York University
jz5348@nyu.edu

Boyuan Chen
New York University
boyuan.chen@nyu.edu

1 Project Overview

1.1 Description

This project aims at designing and building a database system for Alibaba E-Commerce dataset published on Kaggle [1]. We want to provide services such as general order inquires as well as business analytics. The dataset includes 15 .txt file, taking 10.43KB in total. It can be freely accessed via this link.

1.2 Entity sets

- Vendors: vendor ID(p.k.), vendor name, vendor phone, vendor email.
- Category: category ID(p.k.), category name.
- Product: product ID(p.k.), product name, product description.
- Orders: order ID(p.k.), processing fee, shipping fee, tax, total item, order date, delivery status, delivery date, ship name, tracking number, ship address.
- Users: user ID(p.k.), username, password, full name, address, email, phone.
- Credit Card(weak entity to Users): credit card ID (p.k. in conjunction with Users.userID).
- Option: option ID(p.k.), option name.

1.3 Relationship sets

- Product are sold by vendors.
- Products belong to categories.
- Options are associated with products.
- Options and products are contained in carts which is further described by shopping cart ID and quantity.
- Orders has products, which is further described by quantity.
- Orders are placed by users.
- Orders may be paid via credit card.
- Credit card belongs to users.

1.4 Constraints

- Each product belong to exactly one category.
- All products are associated with some options. Each option that exists in the database must be associated with exactly one product.
- All orders must have at least one product.
- Orders cannot be placed by more than one user. If an user is deleted from the database, then user will be set to NULL.
- Credit card information cannot exist independently of its perspective user.

1.5 Business insights

We begin with a routine query which provides us with details on the pruchase history of customer specified by joining orders details and associating the respective user information:

```
SELECT u.full_name            customer,
       o3.option_name         product_option,
       o3.product_id          product_id,
       o2.quantity            quantity,
       o3.price               price,
       o2.quantity * o3.price total_cost
FROM   orders_placed_user o,
       order_has_product o2,
       users u,
       options_associated_with o3
WHERE  o.user_id = u.user_id
       AND o2.order_id = o.order_id
       AND o3.product_id = o2.product_id
       AND u.full_name = '{customer_name}';
```

To a similar effect, it might also be helpful for the E-commerce platform to query sales conducted and visualize the trends as well as build expectations on future business strategies and cash flow management. We again utilize "join" clause to retrieve sales details and proceed to group the query by order date:

```
WITH summary
     AS (SELECT o1.order_date,
                u.full_name            customer,
                o3.option_name         product_option,
                o3.product_id          product_id,
                o2.quantity            quantity,
                o3.price               price,
                o2.quantity * o3.price total_cost
         FROM   orders_placed_user o,
                orders o1,
                order_has_product o2,
                users u,
                options_associated_with o3
         WHERE  o.user_id = u.user_id
                AND o1.order_id = o2.order_id
                AND o2.order_id = o.order_id
                AND o3.product_id = o2.product_id)
SELECT order_date      date,
       Sum(total_cost) revenu
FROM   summary
GROUP  BY order_date
ORDER  BY date
```

To expand on the above breakdown, vendors might also be interested to know its sales ordered by date as part of daily operations to meet sales targets and reconcile to accounting records:

```
WITH vendor_info
     AS (SELECT *
         FROM   vendor
                JOIN product_sold_vendor
                  ON vendor.vendor_id = product_sold_vendor.vendor_id),
     summary
     AS (SELECT o1.order_date,
                u.full_name            customer,
                o3.option_name         product_option,
                o3.product_id          product_id,
```

```
                o2.quantity            quantity,
                o3.price               price,
                o2.quantity * o3.price total_cost
         FROM   orders_placed_user o,
                orders o1,
                order_has_product o2,
                users u,
                options_associated_with o3
         WHERE  o.user_id = u.user_id
                AND o1.order_id = o2.order_id
                AND o2.order_id = o.order_id
                AND o3.product_id = o2.product_id)
SELECT vendor_name,
       order_date,
       Sum(total_cost) revenue
FROM   summary
       JOIN vendor_info
         ON summary.product_id = vendor_info.product_id
WHERE  vendor_name = '{vendor_name}'
GROUP  BY vendor_name,
          order_date
ORDER  BY order_date
```

Costs associated with losing a current customer always outweighs the benefit of acquiring a new one. Among the customers, the delivery status of those who have made more than one purchase can be query after imposing a "having" clause after retrieving the historical orders from customers:

```
WITH repeating_customers
     AS (SELECT ship_name customer
         FROM   orders
         GROUP  BY customer
         HAVING Count(*) > 1)
SELECT order_id,
       ship_name AS customer,
       order_date,
       tracking_number
FROM   orders
WHERE  delivery_status = Cast(0 AS BIT)
       AND ship_name IN (SELECT *
                         FROM   repeating_customers)
```

And finally, E-commerce platforms such as Amazon may utilize information on which products are repeatedly bought. Amazon introduced 'subscript and save' in 2007, which generated expectations of customer pre-orders which may translate to stable cash flows for vendors in exchange for discounts for the consumers. The query below provides details on recurring purchases broken down by customers and product:

```
WITH summary
     AS (SELECT o1.order_id,
                o1.order_date,
                o1.ship_name,
                oo1.product_id,
                oo1.option_id,
                oo1.quantity,
                o1.delivery_status
         FROM   orders o1
                JOIN order_has_product oo1
                  ON o1.order_id = oo1.order_id),
     return_customers
     AS (SELECT s1.order_id,
                s1.order_date,
```

```
                s1.ship_name,
                s1.product_id,
                s1.option_id,
                s1.quantity,
                s2.delivery_status
         FROM   summary s1
                JOIN summary s2
                  ON s1.ship_name = s2.ship_name
                     AND s1.option_id = s2.option_id
         WHERE  s1.order_id <> s2.order_id
                AND s1.order_date < s2.order_date)
SELECT r.ship_name customer,
       o.option_name
FROM   return_customers r
       JOIN options_associated_with o
         ON r.option_id = o.option_id
```

2 Database Schema

2.1 Data Loading

Data loading is implemented as INSERT statement in schema.sql and via the SQL playground implemented as part of the front-end. Please refer to schema.sql for the INSERT statements - excluded for readability and neatness.

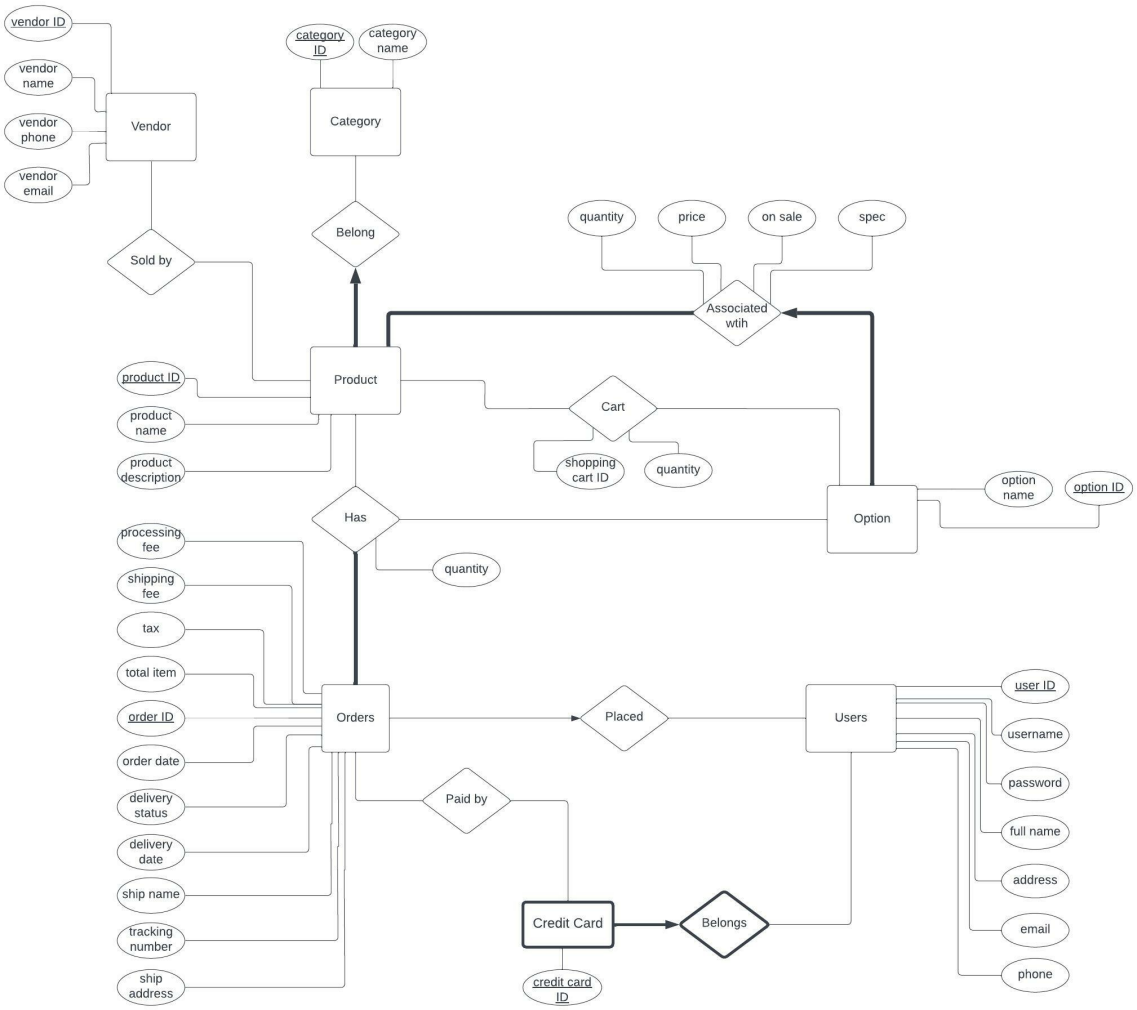
2.2 Schema

```
--> Vendor
CREATE TABLE vendor
  (
     vendor_id    INT PRIMARY KEY,
     vendor_name  VARCHAR(50),
     vendor_phone VARCHAR(20),
     vender_email VARCHAR(50)
  );

--> Category
CREATE TABLE category
  (
     category_id   INT PRIMARY KEY,
     category_name VARCHAR(50)
  );

--> Product_belong
CREATE TABLE product_belong
  (
     product_id   INT PRIMARY KEY,
     product_name VARCHAR(200),
     description  VARCHAR(1000),
     category_id  INT NOT NULL,
     --> key + participation constrain between entity & belong relationship.
     FOREIGN KEY (category_id) REFERENCES category(category_id)
  );

--> Product_sold_vendor
CREATE TABLE product_sold_vendor
  (
     vendor_id  INT,
```

```
    product_id INT,
    PRIMARY KEY (vendor_id, product_id),
    FOREIGN KEY (vendor_id) REFERENCES vendor(vendor_id),
    FOREIGN KEY (product_id) REFERENCES product_belong(product_id)
  );

--> Users
CREATE TABLE users
  (
    user_id   INT PRIMARY KEY,
    username  VARCHAR(20),
    password  VARCHAR(20),
    full_name VARCHAR(30),
    address   VARCHAR(100),
    email     VARCHAR(30),
    phone     VARCHAR(20)
  );

--> Carts_has_products_options
CREATE TABLE carts_has_products_options
  (
    shopping_cart_id INT,
    quantity         INT,
    product_id       INT,
    option_id        INT,
    PRIMARY KEY (shopping_cart_id, product_id, option_id, quantity)
  );

--> Options_associated_with (we cannot represent the the participation constraint
--> from product to options via "associated with" relationship.)
CREATE TABLE options_associated_with
  (
    option_id   INT PRIMARY KEY,
    option_name VARCHAR(300),
    product_id  INT NOT NULL,
    --> to accomodate the key + participation constraint
    --> between option and relationship associated with to products,
    --> we combine "associated with" and options, and bring in product_id.
    quantity    INT,
    price       INT,
    on_sale     BIT,
    specs       VARCHAR(300),
    FOREIGN KEY (product_id) REFERENCES product_belong(product_id)
  );

--> Orders
CREATE TABLE orders
  (
    order_id        INT PRIMARY KEY,
    total_item      INT,
    shipping_fee    INT,
    tax             FLOAT,
    processing_fee  FLOAT,
    order_date      DATE,
    delivery_date   DATE,
    ship_name       VARCHAR(30),
    ship_address    VARCHAR(60),
    tracking_number VARCHAR(10),
    delivery_status BIT
```

```
  );

--> Order_has_Product (participation constrain between order_has_product and orders
--> cannot be enforced without key constraint)
CREATE TABLE order_has_product
  (
     order_id   INT,
     product_id INT,
     option_id  INT,
     quantity   INT,
     PRIMARY KEY (order_id, option_id, product_id),
     FOREIGN KEY (order_id) REFERENCES orders(order_id),
     FOREIGN KEY (product_id) REFERENCES product_belong(product_id),
     FOREIGN KEY (option_id) REFERENCES options_associated_with(option_id)
  );

--> Orders_placed_user
CREATE TABLE orders_placed_user
  (
     user_id  INT,
     order_id INT,
     PRIMARY KEY(user_id, order_id)
  );

--> User_has_creditcard
CREATE TABLE user_has_creditcard
  (
     credit_card_number BIGINT,
     user_id            INT,
     PRIMARY KEY(credit_card_number, user_id),
     FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE
  -- enforced given it is a weak entity
  );

--> Order_paid_CreditCard
CREATE TABLE orders_paid_creditcard
  (
     order_id           INT,
     credit_card_number BIGINT,
     user_id            INT,
     PRIMARY KEY (order_id, credit_card_number, user_id),
     --> given User_has_creditcard is a weak entity,
     --> its parent table Users' user_id is also required as part of the primary key
     FOREIGN KEY (order_id) REFERENCES orders(order_id),
     FOREIGN KEY (credit_card_number, user_id) REFERENCES user_has_creditcard(
     credit_card_number, user_id)
  );
```

3 ER-Diagram



References

[1] Alibaba Group. E-commerce public dataset by alibaba, 2019.