**Database Lab Session #5 Report**

**Group 5**

**DDL**

CREATE DATABASE IF NOT EXISTS lab5;

USE lab5;

CREATE TABLE IF NOT EXISTS HelpdeskInfo (

category VARCHAR(128) PRIMARY KEY,

helpdesk\_email VARCHAR(128)

);

CREATE TABLE IF NOT EXISTS User (

id INT PRIMARY KEY AUTO\_INCREMENT,

username VARCHAR(128) UNIQUE NOT NULL,

email VARCHAR(128) UNIQUE NOT NULL CHECK(email LIKE '%@%.%'),

password VARCHAR(256) NOT NULL CHECK(LENGTH(password ) >= 8),

name VARCHAR(128) NOT NULL,

address VARCHAR(256),

phone\_number VARCHAR(32) CHECK(phone\_number LIKE '%-%-%'),

category VARCHAR(128),

FOREIGN KEY (category) REFERENCES HelpdeskInfo(category) ON DELETE SET NULL ON UPDATE CASCADE

);

CREATE TABLE IF NOT EXISTS Bank (

bank\_code VARCHAR(32) PRIMARY KEY,

bank VARCHAR(128)

);

CREATE TABLE IF NOT EXISTS CreditCard (

number INT PRIMARY KEY AUTO\_INCREMENT,

bank\_code VARCHAR(32),

expire\_date TIMESTAMP DEFAULT NOW(),

user\_id INT NOT NULL,

FOREIGN KEY (bank\_code) REFERENCES Bank(bank\_code) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (user\_id) REFERENCES User(id) ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE IF NOT EXISTS Shop (

id INT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(128) UNIQUE NOT NULL

);

CREATE TABLE IF NOT EXISTS ProductType (

id INT PRIMARY KEY AUTO\_INCREMENT,

description VARCHAR(256),

parent\_id INT,

FOREIGN KEY (parent\_id) REFERENCES ProductType(id) ON DELETE SET NULL ON UPDATE CASCADE

);

CREATE TABLE IF NOT EXISTS Product (

id INT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(128),

color VARCHAR(128),

size VARCHAR(32),

price INT,

description VARCHAR(256),

shop\_id INT NOT NULL,

product\_type INT NOT NULL,

FOREIGN KEY (shop\_id) REFERENCES Shop(id) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (product\_type) REFERENCES ProductType(id) ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE IF NOT EXISTS Photo (

id INT,

photo BLOB,

Product\_id INT NOT NULL,

FOREIGN KEY (product\_id) REFERENCES Product(id),

PRIMARY KEY (id)

);

CREATE TABLE IF NOT EXISTS RestrictedShop (

id INT PRIMARY KEY AUTO\_INCREMENT

);

CREATE TABLE IF NOT EXISTS Restriction (

restricted\_shop INT,

product\_type INT,

FOREIGN KEY (restricted\_shop) REFERENCES RestrictedShop(id) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (product\_type) REFERENCES ProductType(id) ON DELETE CASCADE ON UPDATE CASCADE,

PRIMARY KEY (restricted\_shop, product\_type)

);

CREATE TABLE IF NOT EXISTS OrderTable (

id INT PRIMARY KEY AUTO\_INCREMENT,

date TIMESTAMP DEFAULT NOW(),

status VARCHAR(128) DEFAULT('processing') CHECK(status IN ('processing', 'completed', 'canceled')),

user\_id INT NOT NULL,

FOREIGN KEY (user\_id) REFERENCES User(id) ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE IF NOT EXISTS Shipment (

id INT PRIMARY KEY AUTO\_INCREMENT,

tracking\_number INT,

date TIMESTAMP DEFAULT NOW()

);

CREATE TABLE IF NOT EXISTS OrderItem (

seq\_id INT,

order\_id INT,

product\_id INT,

shipment\_id INT,

unit\_price INT,

quantity INT,

status VARCHAR(128) DEFAULT('processing') CHECK(status IN ('processing', 'shipped', 'out of stock')),

FOREIGN KEY (order\_id) REFERENCES OrderTable(id) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (product\_id) REFERENCES Product(id) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (shipment\_id) REFERENCES Shipment(id) ON DELETE CASCADE ON UPDATE CASCADE,

PRIMARY KEY (seq\_id, order\_id)

);

CREATE TABLE IF NOT EXISTS Invoice (

number INT PRIMARY KEY AUTO\_INCREMENT,

status VARCHAR(128) DEFAULT('issued') CHECK(status IN ('issued', 'paid')),

date TIMESTAMP DEFAULT NOW(),

order\_id INT NOT NULL,

FOREIGN KEY (order\_id) REFERENCES OrderTable(id) ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE IF NOT EXISTS PaymentMethod (

method VARCHAR(128) PRIMARY KEY,

transaction\_fee DOUBLE

);

CREATE TABLE IF NOT EXISTS PaymentID (

id INT PRIMARY KEY AUTO\_INCREMENT,

date TIMESTAMP DEFAULT NOW(),

amount INT,

invoice\_number INT NOT NULL,

creditcard\_number INT NOT NULL,

method VARCHAR(128) NOT NULL,

FOREIGN KEY (invoice\_number) REFERENCES Invoice(number) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (creditcard\_number) REFERENCES CreditCard(number) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (method) REFERENCES PaymentMethod(method) ON DELETE CASCADE ON UPDATE CASCADE

);

**DDL Description**

The above DDL will create the database ‘lab5’ and the tables we defined in Lab Session #3. The detailed explanations of the tables are as follows.

* HelpdeskInfo
* User
  + Since username and email are unique, we specified UNIQUE and NOT NULL constraints
  + For email, we added simple attribute based check for detection of the typical email format ([example@domain.com](mailto:example@domain.com)) with the pattern “%@%.%”
  + We assumed that the password should be longer than or equal to 8 characters. So we used an attribute based check for checking it
  + We used attribute based check for checking the format of the phone number, which should be the general format of the Korean phone number 010-xxxx-xxxx
  + We assumed that the password and name should be specified for a user so we used NOT NULL constraint to force them
  + Category field is the foreign key for referencing HelpdeskInfo table
  + We set ON DELETE action for category field to SET NULL to prevent the deletion of user records when a category is removed by mistake or temporarily removed
* Bank
* CreditCard
  + Since every credit card has its owner id, we set the user\_id as NOT NULL
  + bank\_code field is the foreign key for referencing Bank table
  + user\_id field is the foreign key for referencing User table
* Shop
  + We set the name field to UNIQUE and NOT NULL according to the specification on the description
* ProductType
  + parent\_id field is the foreign key referencing itself. It specifies the parent product type id, which can be referenced recursively, forming hierarchy of the types
  + We set the ON DELETE action of the parent\_id field to SET NULL since the sub type can become a root type after the deletion of the parent type
* Product
  + shop\_id field is the foreign key referencing Shop table.
  + Since every product has shop info, we set this field to NOT NULL
  + product\_type field is the foreign key referencing ProductType table
  + Since every product has its type, we set this field to NOT NULL
* Photo
  + We specified the type of the photo field to BLOB, which represents the binary data in MySQL
  + Because the BLOB type can’t be used in PK, and id field itself is sufficient to identify each record, we set only id field as PK for Photo table
  + product\_id field is the foreign key referencing Product table
  + Since every photo has the product info it belongs to, we set this field to NOT NULL
* RestrictedShop
  + This table contains the ids of the shops need to restrict the selling of a product with specific type
* Restriction
  + restricted\_shop field is the foreign key referencing the RestrictedShop table
  + product\_type field is the foreign key referencing the ProductType table
  + The above two foreign keys are the PK of this table
* OrderTable
  + Since MySQL didn’t allow to use the name “Order” as this table, we set the name of this table as “OrderTable”
  + We added an attribute based check to the status field to force it to be one of the values among “processing”, “completed”, and “canceled” as written in the description
  + user\_id field is the foreign key referencing the User table and should be NOT NULL
* Shipment
* OrderItem
  + order\_id is the foreign key referencing the OrderTable table
  + product\_id is the foreign key referencing the Product table
  + shipment\_id is the foreign key referencing the Shipment table
  + The seq\_id and order\_id are the PK of this table
* Invoice
  + We added an attribute based check to the status field to force it to be one of the values between “issued” and “paid” as written in the description
  + order\_id is the foreign key referencing the OrderTable table and should be NOT NULL
* PaymentMethod
* PaymentID
  + method is the foreign key referencing the PaymentMethod table and should be NOT NULL
  + creditcard\_number is the foreign key referencing the CreditCard table and should be NOT NULL
  + invoice\_number is the foreign key referencing the Invoice table and should be NOT NULL

**Dummy Data**

# Helpdesk Information

INSERT INTO HelpdeskInfo(category, helpdesk\_email) VALUES

('regular', 'servicedesk1@company.com'),

('premium', 'servicedesk2@company.com');

# Users

INSERT INTO User(username, email, password, name, address, phone\_number, category) VALUES

('user1', 'user1@email.com', 'password1', 'John Doe', '서울특별시 강남구 도곡동 123번지', '010-1111-2222', 'regular'),

('user2', 'user2@email.com', 'password2', 'Jane Smith', '서울특별시 서초구 양재동 456번지', '010-2222-3333', 'premium'),

('user3', 'user3@email.com', 'password3', 'Bob Johnson', '서울특별시 송파구 잠실동 789번지', '010-3333-4444', 'regular'),

('user4', 'user4@email.com', 'password4', 'Alice Brown', '서울특별시 강북구 미아동 101번지', '010-4444-5555', 'premium'),

('user5', 'user5@email.com', 'password5', 'Charlie Davis', '서울특별시 강서구 화곡동 202번지', '010-5555-6666', 'regular'),

('user6', 'user6@email.com', 'password6', 'Diana White', '서울특별시 중랑구 면목동 303번지', '010-6666-7777', 'premium'),

('user7', 'user7@email.com', 'password7', 'Edward Miller', '서울특별시 강동구 천호동 404번지', '010-7777-8888', 'regular'),

('user8', 'user8@email.com', 'password8', 'Fiona Wilson', '서울특별시 강남구 삼성동 505번지', '010-8888-9999', 'premium'),

('user9', 'user9@email.com', 'password9', 'George Taylor', '서울특별시 서대문구 홍제동 606번지', '010-9999-0000', 'regular'),

('user10', 'user10@email.com', 'password10', 'Helen Adams', '서울특별시 강서구 방화동 707번지', '010-0000-1111', 'premium'),

('user11', 'user11@email.com', 'password11', 'Ella Lee', '서울특별시 강남구 역삼동 1111번지', '010-1111-2222', 'regular'),

('user12', 'user12@email.com', 'password12', 'Benjamin Kim', '서울특별시 송파구 거여동 1222번지', '010-2222-3333', 'premium'),

('user13', 'user13@email.com', 'password13', 'Mia Park', '서울특별시 강서구 방화동 1333번지', '010-3333-4444', 'regular'),

('user14', 'user14@email.com', 'password14', 'Sebastian Choi', '서울특별시 강북구 미아동 1444번지', '010-4444-5555', 'premium'),

('user15', 'user15@email.com', 'password15', 'Amelia Kang', '서울특별시 중랑구 면목동 1555번지', '010-5555-6666', 'regular'),

('user16', 'user16@email.com', 'password16', 'Henry Ryu', '서울특별시 강동구 천호동 1666번지', '010-6666-7777', 'premium'),

('user17', 'user17@email.com', 'password17', 'Evelyn Park', '서울특별시 강남구 삼성동 1777번지', '010-7777-8888', 'regular'),

('user18', 'user18@email.com', 'password18', 'Matthew Kim', '서울특별시 강서구 화곡동 1888번지', '010-8888-9999', 'premium'),

('user19', 'user19@email.com', 'password19', 'Stella Lee', '서울특별시 서대문구 홍제동 1999번지', '010-9999-0000', 'regular'),

('user20', 'user20@email.com', 'password20', 'Leo Park', '서울특별시 강동구 강일동 2010번지', '010-0000-1111', 'premium'),

('user21', 'user21@email.com', 'password21', 'Chloe Kim', '서울특별시 강북구 수유동 2111번지', '010-1111-2222', 'regular'),

('user22', 'user22@email.com', 'password22', 'James Lee', '서울특별시 송파구 잠실동 2222번지', '010-2222-3333', 'premium'),

('user23', 'user23@email.com', 'password23', 'Lily Park', '서울특별시 강서구 마곡동 2333번지', '010-3333-4444', 'regular'),

('user24', 'user24@email.com', 'password24', 'Daniel Kim', '서울특별시 강남구 대치동 2444번지', '010-4444-5555', 'premium'),

('user25', 'user25@email.com', 'password25', 'Avery Lee', '서울특별시 강동구 상일동 2555번지', '010-5555-6666', 'regular'),

('user26', 'user26@email.com', 'password26', 'Jackson Park', '서울특별시 중랑구 면목동 2666번지', '010-6666-7777', 'premium'),

('user27', 'user27@email.com', 'password27', 'Scarlett Kim', '서울특별시 강남구 압구정동 2777번지', '010-7777-8888', 'regular'),

('user28', 'user28@email.com', 'password28', 'Logan Lee', '서울특별시 강서구 등촌동 2888번지', '010-8888-9999', 'premium'),

('user29', 'user29@email.com', 'password29', 'Zoe Park', '서울특별시 서대문구 연희동 2999번지', '010-9999-0000', 'regular'),

('user30', 'user30@email.com', 'password30', 'Carter Kim', '서울특별시 강동구 천호동 3010번지', '010-0000-1111', 'premium'),

('user31', 'user31@email.com', 'password31', 'Madison Lee', '서울특별시 강북구 미아동 3111번지', '010-1111-2222', 'regular'),

('user32', 'user32@email.com', 'password32', 'Lucas Park', '서울특별시 송파구 가락동 3222번지', '010-2222-3333', 'premium'),

('user33', 'user33@email.com', 'password33', 'Aria Kim', '서울특별시 강서구 등촌동 3333번지', '010-3333-4444', 'regular'),

('user34', 'user34@email.com', 'password34', 'Jack Lee', '서울특별시 강남구 신사동 3444번지', '010-4444-5555', 'premium'),

('user35', 'user35@email.com', 'password35', 'Hazel Park', '서울특별시 중랑구 망우동 3555번지', '010-5555-6666', 'regular'),

('user36', 'user36@email.com', 'password36', 'Elijah Kim', '서울특별시 강동구 상일동 3666번지', '010-6666-7777', 'premium'),

('user37', 'user37@email.com', 'password37', 'Luna Lee', '서울특별시 강남구 역삼동 3777번지', '010-7777-8888', 'regular'),

('user38', 'user38@email.com', 'password38', 'Oliver Park', '서울특별시 강서구 마곡동 3888번지', '010-8888-9999', 'premium'),

('user39', 'user39@email.com', 'password39', 'Layla Kim', '서울특별시 서대문구 홍제동 3999번지', '010-9999-0000', 'regular'),

('user40', 'user40@email.com', 'password40', 'Jacob Lee', '서울특별시 강동구 강일동 4010번지', '010-0000-1111', 'premium');

# Banks

INSERT INTO Bank(bank\_code, bank) VALUES

('01', 'KB국민은행'),

('02', '우리은행'),

('03', 'IBK기업은행'),

('04', 'NH농협은행'),

('05', '한국씨티은행'),

('06', '신한은행'),

('07', 'SC제일은행'),

('08', '대구은행'),

('09', '부산은행'),

('10', '광주은행'),

('11', '제주은행'),

('12', '전북은행'),

('13', '경남은행'),

('14', '새마을금고'),

('15', '신협'),

('16', '상호저축은행'),

('17', '한국씨티은행'),

('18', '수협은행'),

('19', '카카오뱅크'),

('20', '케이뱅크');

# Credit Cards

INSERT INTO CreditCard(bank\_code, expire\_date, user\_id) VALUES

('01', '2025-01-14', 1),

('05', '2027-03-08', 1),

('09', '2026-05-22', 2),

('11', '2025-07-16', 2),

('13', '2027-09-10', 3),

('19', '2025-01-14', 4),

('11', '2025-07-16', 5),

('15', '2027-09-10', 6),

('11', '2027-03-08', 13),

('13', '2026-05-22', 14),

('15', '2025-07-16', 14),

('17', '2027-09-10', 15),

('19', '2026-11-04', 15),

('02', '2025-01-14', 16),

('04', '2027-03-08', 16),

('08', '2025-07-16', 17),

('10', '2027-09-10', 18),

('12', '2026-11-04', 18),

('14', '2025-01-14', 19),

('16', '2027-03-22', 19),

('13', '2027-09-10', 24),

('15', '2026-11-04', 24),

('17', '2025-01-14', 25),

('19', '2027-03-22', 25),

('02', '2026-05-22', 26),

('04', '2025-07-16', 26),

('06', '2027-09-10', 27),

('08', '2026-11-04', 27),

('19', '2025-07-16', 35),

('02', '2027-09-10', 36),

('04', '2026-11-04', 36),

('06', '2025-01-14', 37);

# Shops

INSERT INTO Shop(name) VALUES

('shop1'),

('shop2'),

('shop3'),

('shop4'),

('shop5');

# Product Types

INSERT INTO ProductType(description, parent\_id) VALUES

('Electronics', NULL),

('Laptops', 1),

('Smartphones', 1),

('Headphones', 1),

('Clothing', NULL),

('Men''s Clothing', 5),

('Women''s Clothing', 5),

('Footwear', 5),

('Books', NULL),

('Fiction', 9),

('Non-Fiction', 9),

('Science Fiction', 10),

('Mystery', 11),

('History', 11),

('Home and Kitchen', NULL),

('Appliances', 15),

('Cookware', 15),

('Furniture', 15),

('Toys and Games', NULL),

('Board Games', 19),

('Outdoor Toys', 19),

('Action Figures', 19),

('Sports and Outdoors', NULL),

('Outdoor Recreation', 23),

('Athletic Clothing', 23),

('Exercise Equipment', 23),

('Beauty and Personal Care', NULL),

('Skincare', 27),

('Haircare', 27),

('Makeup', 27);

# Products

INSERT INTO Product(name, color, size, price, description, shop\_id, product\_type) VALUES

('Laptop X1', 'Silver', '15 inches', 1540000, 'Powerful laptop with high-performance features', 1, 2),

('Smartphone S1', 'Black', '6.2 inches', 780000, 'Latest smartphone with advanced features', 1, 3),

('Wireless Headphones H2', 'Blue', NULL, 175000, 'Bluetooth headphones with noise cancellation', 1, 4),

('Men''s T-Shirt', 'Red', 'Medium', 32000, 'Comfortable cotton t-shirt for men', 2, 6),

('Women''s Dress', 'Yellow', 'Small', 56000, 'Elegant dress for women for any occasion', 2, 7),

('Running Shoes', 'White', '9', 88000, 'Lightweight and comfortable shoes for running', 2, 8),

('Science Fiction Book', NULL, NULL, 24900, 'Exciting science fiction novel by author XYZ', 3, 12),

('Science Fiction Book2', NULL, NULL, 24900, 'Exciting science fiction novel by author XYZ', 3, 12),

('Science Fiction Book3', NULL, NULL, 24900, 'Exciting science fiction novel by author XYZ', 3, 12),

('Science Fiction Book4', NULL, NULL, 24900, 'Exciting science fiction novel by author XYZ', 3, 12),

('History Book', NULL, NULL, 24900, 'About world''s history', 3, 14),

('History Book2', NULL, NULL, 24900, 'About world''s history2', 3, 14),

('Fiction book', NULL, NULL, 24900, 'Fiction story', 3, 10),

('Cookware Set', 'Stainless Steel', NULL, 163000, 'High-quality cookware set for your kitchen', 3, 17),

('Cookware Set2', 'Stainless Steel', NULL, 163000, 'High-quality cookware set for your kitchen', 3, 17),

('Board Game: Mystery Mansion', 'Multicolor', NULL, 44000, 'Engaging mystery board game for the whole family', 4, 20),

('Outdoor Sports Backpack', 'Green', 'Large', 54000, 'Durable backpack for outdoor activities', 4, 26),

('Skincare Set', NULL, NULL, 66000, 'Complete skincare set for a radiant complexion', 5, 28),

('Exercise Bike', 'Black', NULL, 299000, 'Indoor exercise bike for a great workout', 5, 26);

# Photos

INSERT INTO Photo(id, photo, product\_id) VALUES

(1, 1234111, 1),

(2, 1351151, 1),

(3, 2636262, 2);

# RestrictedShop

INSERT INTO RestrictedShop(id) VALUES

(1),

(2),

(3);

# Restrictions

INSERT INTO Restriction(restricted\_shop,product\_type) VALUES

(1, 8), # Footware

(2, 3), # Smartphones

(3, 28); # Skincare

# OrderTable

INSERT INTO OrderTable (date, status, user\_id) VALUES

('2023-12-01 08:00:00','processing', 1),

('2023-12-02 10:30:00','completed', 2),

('2023-12-03 15:45:00','canceled', 3);

# Shipment

INSERT INTO Shipment (tracking\_number, date) VALUES

(123456789, '2023-12-01 08:00:00'),

(987654321, '2023-12-02 10:30:00'),

(555555555, '2023-12-03 15:45:00');

# OrderItem

INSERT INTO OrderItem (seq\_id, order\_id, product\_id, shipment\_id, unit\_price, quantity, status) VALUES

(1, 1, 3, NULL, 30, 3, 'processing'),

(2, 1, 1, 1, 50, 1, 'shipped'),

(3, 1, 4, 1, 40, 2, 'shipped'),

(4, 1, 1, NULL, 70, 3, 'processing'),

(1, 2, 5, 2, 75, 3, 'shipped'),

(2, 2, 9, NULL, 55, 4, 'processing'),

(3, 2, 2, NULL, 50, 7, 'processing'),

(1, 3, 6, 3, 45, 2, 'shipped'),

(2, 3, 5, NULL, 100, 1, 'out of stock');

# Invoice

INSERT INTO Invoice (status, date, order\_id) VALUES

('paid', '2023-12-01 08:00:00', 1),

('paid', '2023-12-02 10:30:00', 2),

('issued', '2023-12-03 15:45:00', 3);

# PaymentMethod

INSERT INTO PaymentMethod (method, transaction\_fee) VALUES

('Credit Card', 0.02),

('PayPal', 0.03),

('Bank Transfer', 0.01);

# PaymentID

INSERT INTO PaymentID (date, amount, invoice\_number, creditcard\_number, method) VALUES

('2023-11-01 08:00:00', 430, 1, '1', 'Credit Card'),

('2023-11-02 10:30:00', 795, 2, '2', 'PayPal');

**Sample Queries**

1. Using a customer's email address, can you provide the IDs of products that this customer has purchased and paid for, but have not been shipped yet?

| SELECT DISTINCT item.product\_id  FROM User u, OrderTable ord, OrderItem item, Invoice inv  WHERE u.id = ord.user\_id  AND item.order\_id = ord.id AND inv.order\_id = ord.id  AND inv.status = 'paid' AND item.status = 'processing'  AND u.email = 'user1@email.com'; |
| --- |

The above query will retrieve all product ids of the order items that are paid but not yet shipped (only include ‘processing’ status) which are ordered by the user with a given email address. At the end of the query, ‘[user1@email.com](mailto:user1@email.com)’ is the temporary email so in real situation it can be an argument passed by the application for retrieving a certain user’s query result.

1. Can you pinpoint the three product type IDs that have the highest sales quantities? Only consider products that have been ordered and paid for, disregarding their shipment status.

| SELECT product\_type  FROM Product p, OrderItem item, Invoice inv  WHERE item.product\_id = p.id AND item.order\_id = inv.order\_id AND inv.status = 'paid'  GROUP BY product\_type ORDER BY SUM(item.quantity) DESC LIMIT 3; |
| --- |

The above query will give the product type with the highest sales quantities only regarding the payment status (status field of Invoice table should be ‘paid’). It groups all the order items by the product\_type, order by its sum of quantities, then picks the top 3 records.

1. Could you list the descriptions of product types that are categorized at the second tier?

| SELECT description FROM ProductType p WHERE p.parent\_id IN (SELECT id FROM ProductType pp WHERE pp.parent\_id IS NULL); |
| --- |

The above query retrieves all the second-tier product types. The method to get all the second-tier types is checking whether its parent type is the first-tier types since it’s easy to check the first-tier types (parent\_id = NULL).

1. Which pair of product IDs are most frequently ordered together?

| SELECT item1.product\_id AS P1, item2.product\_id AS P2, COUNT(\*)  FROM OrderItem item1, OrderItem item2  WHERE item1.order\_id = item2.order\_id AND item1.seq\_id <> item2.seq\_id  GROUP BY item1.product\_id, item2.product\_id ORDER BY COUNT(\*) DESC LIMIT 1; |
| --- |

We thought that the standard of the ‘frequently ordered’ is the higher number of orders containing both product IDs in the pair. So we first get all the combinations of the pair of order items in the same order with different seq\_id (to prevent self-pairing), then group by the combination of the two product\_types in the pair to calculate the sum of such combinations per order. We assumed that each order has no two order items with the same product\_id to prevent the duplicate counting per order.

1. Can you randomly pick three customers and share their email addresses?

| SELECT email FROM User ORDER BY RAND() LIMIT 3; |
| --- |

Rand() is the mysql function that generates a random number. By using this, the above query uses the random number generated per each record as a key for sorting, which leads to the random sort. Then it takes the first 3 records to pick the 3 random emails from the result randomly ordered.

**Additional Queries**

1. Get the number of products in each product type including the number of products in its subtypes

| WITH RECURSIVE NUM\_TYPES AS (  SELECT pt.id, pt.description, pt.parent\_id, COUNT(p.id) AS NumTypes  FROM ProductType pt LEFT OUTER JOIN Product p on pt.id = p.product\_type  GROUP BY pt.id, pt.description, pt.parent\_id  UNION ALL  SELECT pt.id, pt.description, pt.parent\_id, nt.NumTypes  FROM ProductType pt, NUM\_TYPES nt WHERE pt.id = nt.parent\_id  )  SELECT id, description, parent\_id, SUM(NumTypes) AS Total  FROM NUM\_TYPES GROUP BY id, description, parent\_id ORDER BY id; |
| --- |

The above query returns the following result when there are product and product types as follows.

Product

| id | name | product\_type |
| --- | --- | --- |
| 1 | Laptop X1 | 2 |
| 2 | SmartPhone S1 | 3 |
| 4 | Men’s T-Shirt | 4 |

ProductType

| id | description | parent\_id |
| --- | --- | --- |
| 1 | Electronics | NULL |
| 2 | Laptops | 1 |
| 3 | SmartPhones | 1 |
| 5 | Clothing | NULL |
| 6 | Men’s Clothing | 5 |

Query Result

| id | description | parent\_id | Total |
| --- | --- | --- | --- |
| 1 | Electronics | NULL | 2 |
| 2 | Laptops | 1 | 1 |
| 3 | SmartPhones | 1 | 1 |
| 5 | Clothing | NULL | 1 |
| 6 | Men’s Clothing | 5 | 1 |

The “electronics” type has “Laptops” and “Smartphones” as the children types so when counting the number of products in each product type, it has a total of 2 products belonging to itself. Similarly, “clothing” has a total 1 product since the count of the product in “Men’s clothing” type is 1, which is the child type of “clothing”.

| SELECT pt.id, pt.description, pt.parent\_id, COUNT(p.id) AS NumTypes  FROM ProductType pt LEFT OUTER JOIN Product p on pt.id = p.product\_type  GROUP BY pt.id, pt.description, pt.parent\_id |
| --- |

Initial part of the query

To do this, it uses “WITH RECURSIVE” syntax, which allows the query to be executed recursively by referencing the updated table. It consists of the initial query, which is the upper part divided by “UNION ALL” and the recursive query, which is the below part. In the initial query, it retrieves all the product types and the number of products belonging to the type by joining it with the product.

| SELECT pt.id, pt.description, pt.parent\_id, nt.NumTypes  FROM ProductType pt, NUM\_TYPES nt WHERE pt.id = nt.parent\_id |
| --- |

Recursive part of the query

And then for each recursive query step, it retrieves all the product types that have one of the product types in the current records as parent type. And it unions previous records and newly retrieved types and repeats this step until all the product types are included. The result is as follows.

| id | description | parent\_id | NumTypes |
| --- | --- | --- | --- |
| 1 | Electronics | NULL | 0 |
| 2 | Laptops | 1 | 1 |
| 3 | SmartPhones | 1 | 1 |
| 5 | Clothing | NULL | 0 |
| 6 | Men’s Clothing | 5 | 1 |

Result right after the initial step

| id | description | parent\_id | NumTypes |
| --- | --- | --- | --- |
| 1 | Electronics | NULL | 0 |
| 2 | Laptops | 1 | 1 |
| 1 | Electronics | NULL | 1 |
| 3 | SmartPhones | 1 | 1 |
| 1 | Electronics | NULL | 1 |
| 5 | Clothing | NULL | 0 |
| 6 | Men’s Clothing | 5 | 1 |
| 5 | Clothing | NULL | 1 |

After recursive step

The record with the same color represents that it was derived from the product type that has the other type as parent. After the recursive step, the query groups all the types by the id to get the sum of the number of products (NumTypes).

| SELECT id, description, parent\_id, SUM(NumTypes) AS Total  FROM NUM\_TYPES GROUP BY id, description, parent\_id ORDER BY id; |
| --- |

Aggregation query after the recursive query

| id | description | parent\_id | Total |
| --- | --- | --- | --- |
| 1 | Electronics | NULL | 2 |
| 2 | Laptops | 1 | 1 |
| 3 | SmartPhones | 1 | 1 |
| 5 | Clothing | NULL | 1 |
| 6 | Men’s Clothing | 5 | 1 |

After aggregation

1. User checks the delivery date of their order

We have implemented a tracking feature that can be used when a user wants to know when the item they ordered will arrive. In this example, we have assumed tracking for user1.

| SELECT username, order\_id, prod.name as ProductName, quantity, ord.status as OrderStatus, item.status as ItemStatus, ship.date  FROM OrderTable ord, User usr, OrderItem item, Product prod, Shipment ship  WHERE ord.user\_id = usr.id AND item.order\_id = ord.id AND item.product\_id = prod.id AND item.shipment\_id = ship.id AND usr.id = 1 ORDER BY ord.id; |
| --- |

Conditions that specify relationships between tables:

Match the user ID in the Orders table with the ID in the Users table.

Match the order ID of the OrderItem with the ID in the OrderTable.

Match the OrderItem's product ID with the ID in the Product table.

Match the shipping ID of the OrderItem with the ID in the Shipping table.

Finally, select the information for the user with User ID 1

And finally, we sort the results by the ID in the OrderTable for overlapping results.

| username | order\_id | ProductName | quantity | OrderStatus | ItemStatus | ship.date |
| --- | --- | --- | --- | --- | --- | --- |
| user1 | 1 | Laptop X1 | 1 | processing | processing | 2023-12-01 08:00:00' |

1. Knowing where a product is sold and its price

If we're looking for an item in a mall and you know that it's available at a store near you, you can save time on shipping.

| SELECT p.name as ProductName, s.name as ShopName, price FROM Product p, Shop s WHERE p.shop\_id = s.id; |
| --- |

| ProductName | ShopName | price |
| --- | --- | --- |
| Laptop X1 | shop1 | 1540000 |
| Smartphone S1 | shop1 | 780000 |
| Wireless Headphones H2 | shop1 | 175000 |
| Men''s T-Shirt | shop2 | 32000 |
| Women''s Dress | shop2 | 56000 |
| Running Shoes | shop2 | 88000 |
| Science Fiction Book | shop3 | 24900 |
| Cookware Set | shop3 | 163000 |
| Board Game: Mystery Mansion | shop4 | 44000 |
| Outdoor Sports Backpack | shop4 | 54000 |
| Skincare Set | shop5 | 66000 |
| Exercise Bike | shop5 | 299000 |

1. Check the total amount of payments for a specific period

Many shopping mall sites have implemented similar queries to check the total amount of payments for the past year or a specific period. I used the total amount of payments for the last 6 months as an example to keep a household account of our over-consumption.

| SELECT usr.username, ord.id as orderId, (pi.amount \* (1 + pm.transaction\_fee)) as Paid, pi.method, pi.date  FROM User usr, OrderTable ord, Invoice inv, PaymentId pi, PaymentMethod pm  WHERE ord.user\_id = usr.id AND inv.order\_id = ord.id AND pi.invoice\_number = inv.number  AND pi.method = pm.method AND inv.status = 'paid' AND pi.date >= DATE\_SUB(NOW(), INTERVAL 6 MONTH); |
| --- |

We used the now() function supported by mysql to calculate and use the time as the basis for executing this query.

Find an order that matches the user ID in the OrderTable and the ID in the User table. Based on the found order, check if there is an order in the Invoice. If there is an order, compare the invoice number in the PaymentId with the number in the Invoice. Based on this, we match the payment method with the method in PaymentMethod. After that, we select the invoices with a status of 'paid' that have actually been paid. Finally, we check if the Select when the payment date is within 6 months of the current date.

| username | orderID | Paid | method | date |
| --- | --- | --- | --- | --- |
| user2 | 2 | 154.5 | PayPal | '2023-11-02 10:30:00' |

**Triggers**

1. CreateInvoice

The purpose of this trigger is to create an invoice for an order whenever an order is created. The invoice cannot exist before the order, and should only be generated when the order comes in. By automatically generating invoices for related orders, you will facilitate order tracking and payment processing.

| DROP TRIGGER CreateInvoice;  # Invoice  DELIMITER $$  CREATE TRIGGER CreateInvoice  AFTER INSERT ON OrderTable  FOR EACH ROW  BEGIN  INSERT INTO Invoice(order\_id) VALUES(NEW.id);  END  $$ DELIMITER ; |
| --- |

1. RestrictSell

The purpose of this trigger is to check if the new product is available for sale in a particular store and not sell it if it is restricted. In order for the store to do its job properly, it should only bring in items that are not on the restricted items in the restricted store. So if it turns out to be restricted, we roll back the trigger using SIGNAL SQLSTATE '45000'.

| DROP TRIGGER RestrictSell;  # RestrictedShop  DELIMITER $$  CREATE TRIGGER RestrictSell  BEFORE INSERT ON Product  FOR EACH ROW  BEGIN  IF NEW.product\_type IN (  SELECT DISTINCT rn.product\_type  FROM Restriction rn, RestrictedShop rs  WHERE rn.restricted\_shop = rs.id AND rs.id = NEW.shop\_id) THEN  SIGNAL SQLSTATE '45000';  END IF;  END  $$ DELIMITER ; |
| --- |