DATABASE DESIGN & PROGRAMMING - FINAL PROJECT REPORT

Team 6: Online Shop Application

Members:

Lahiru Madhusanka Hewawasam Halloluwage

Udalmaththa Gamage Chathuri Anuththara Karunarathna

Sasvi Vidunadi Ranasinghe

Himihami Mudiyanselage Lahiru Bandaranayake

Introduction

This project was developed as part of the **Database Design and Programming** course in the second year of the Information and Communication Technology, Robotics degree programme.
The objective was to design and implement a fully functional **relational database system** for an **Online Shop Application**.

The system manages several core business processes such as customer registration, product cataloguing, order placement, payment processing, and shipment tracking.
Our aim was to demonstrate a complete end-to-end database design life cycle, including ER modelling, normalization, schema creation (DDL), sample data (DML), and advanced SQL queries.

The database was implemented using **MySQL** and designed using **MySQL Workbench**, **ERDPlus**, and **Creately** for diagramming.

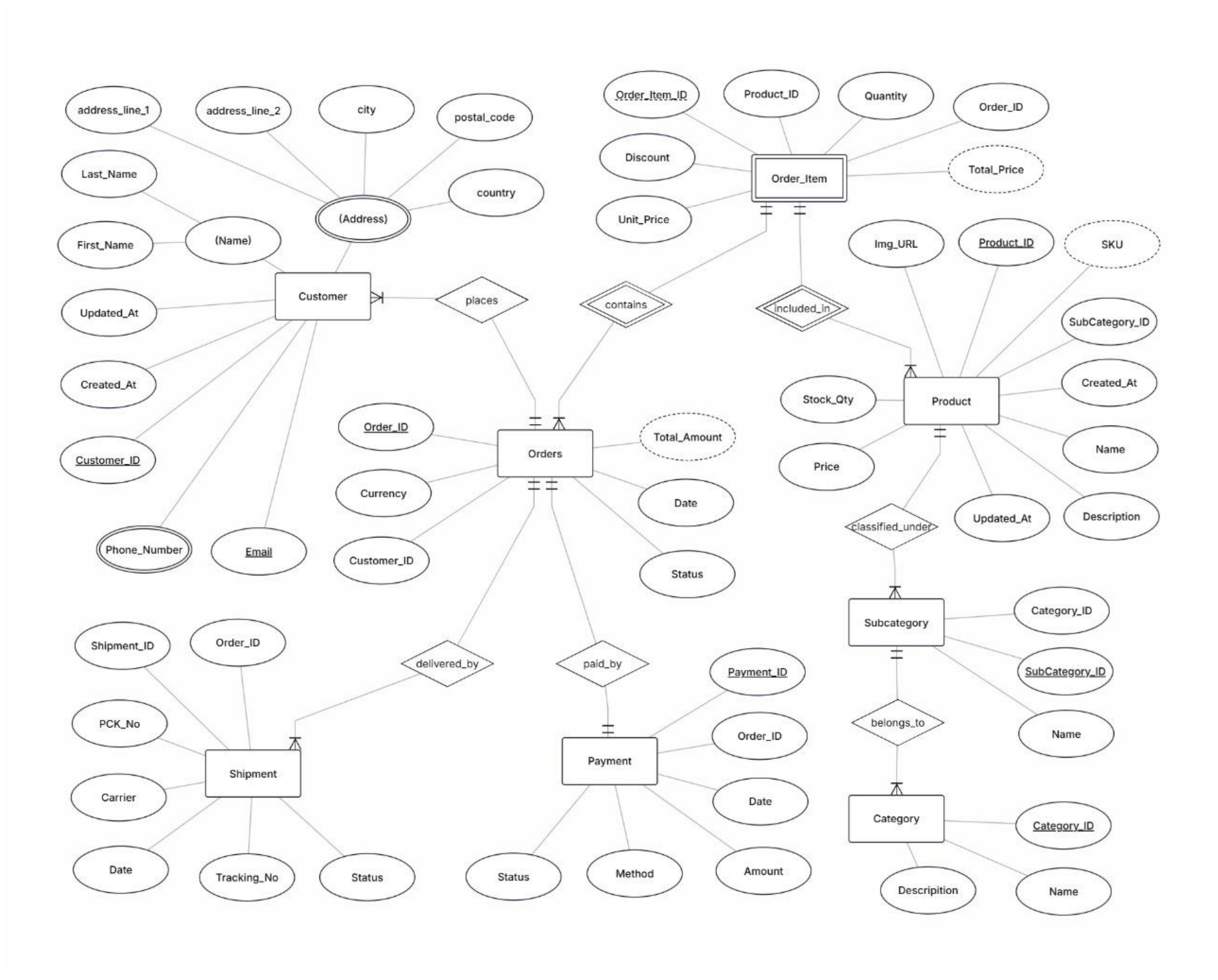
Entity–Relationship (ER) Diagram

The Entity–Relationship (ER) diagram represents the conceptual design of the *Online Shop Application* database.
It identifies the key entities, their attributes, and the relationships between them.
The ER model was designed using MySQL Workbench and ERDPlus before implementing the database.

Key Entities and Relationships

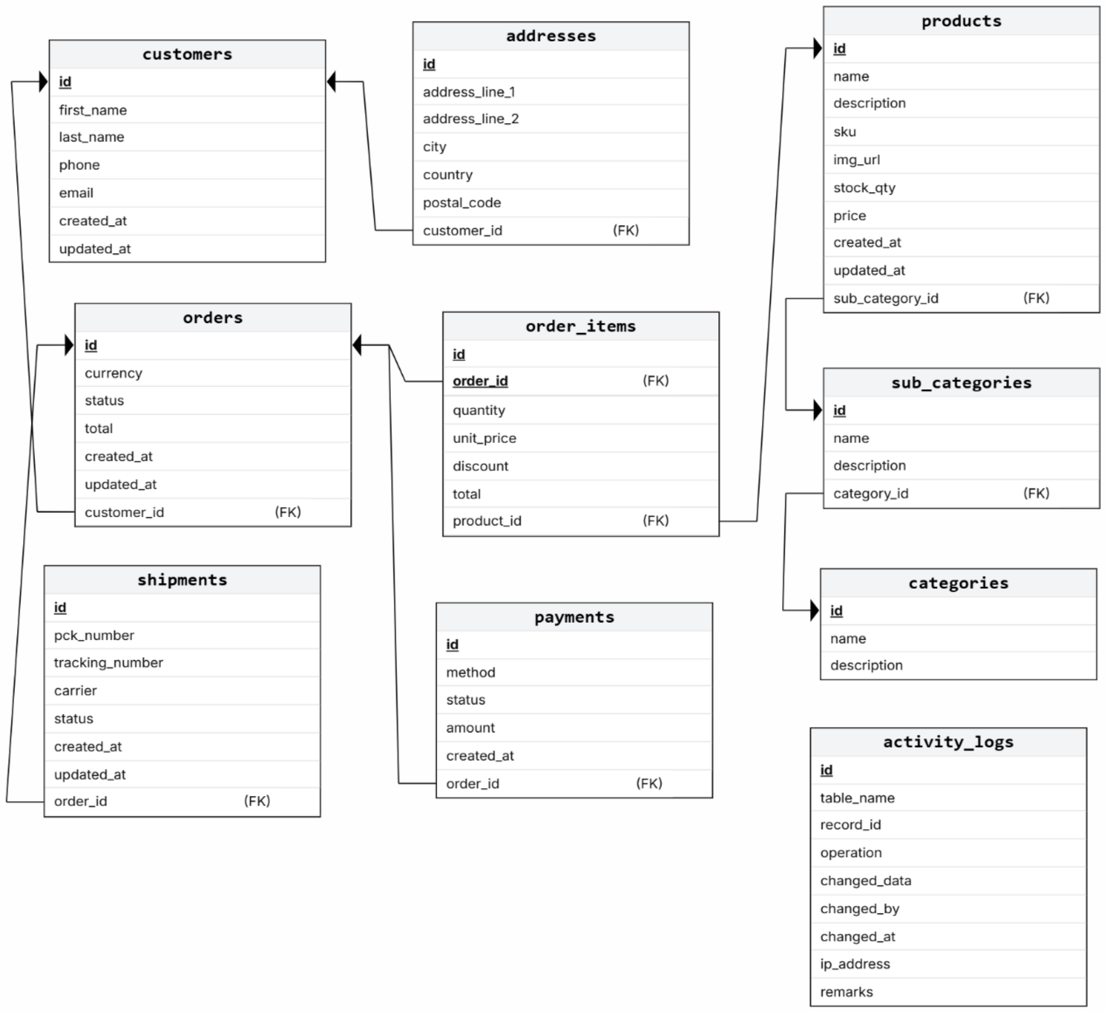
- Customer – Order: One customer can place many orders (1 to *).
- Order – OrderItem: Each order can contain multiple order items (1 to *).
- OrderItem – Product: Each order item corresponds to exactly one product (* to 1).
- Product – SubCategory: Each product belongs to one subcategory (* to 1).
- SubCategory – Category: Each subcategory belongs to one category (* to 1).
- Order – Payment: Each order has one corresponding payment record (1 to 1).
- Order – Shipment: Each order may have one shipment record (1 to 0..1).

The ER diagram ensures referential integrity between entities and provides a clear blueprint for converting the conceptual model into a relational schema.


address_line_1
address_line_2
city
postal_code
country
(Address)
Last_Name
First_Name
(Name)
Updated_At
Created_At
Customer_ID
Phone_Number
Email
Customer
places
Order_Item_ID
Product_ID
Quantity
Order_ID
Discount
Order_Item
Total_Price
Unit_Price
contains
included_in
Img_URL
Product_ID
SKU
SubCategory_ID
Created_At
Stock_Qty
Product
Price
Name
Description
Updated_At
classified_under
Order_ID
Orders
Total_Amount
Currency
Date
Customer_ID
Status
Subcategory
Category_ID
SubCategory_ID
Name
belongs_to
Category
Category_ID
Descripition
Name
Shipment_ID
Order_ID
delivered_by
paid_by
PCK_No
Shipment
Carrier
Date
Tracking_No
Status
Payment_ID
Order_ID
Payment
Date
Status
Method
Amount

Relational Schema (with Keys & Constraints)

Table	**Primary Key**	**Foreign Keys**	**Description**
Customer	Customer_ID	Address_ID	Stores customer details and contact info
Address	Address_ID	–	Physical address data
Order	Order_ID	Customer_ID	Records each purchase transaction
OrderItem	OrderItem_ID	Order_ID, Product_ID	Links orders to purchased products
Product	Product_ID	SubCategory_ID	Product details and stock quantities
Category	Category_ID	–	Main product categories
SubCategory	SubCategory_ID	Category_ID	Subdivision of categories
Payment	Payment_ID	Order_ID	Payment information and status
Shipment	Shipment_ID	Order_ID	Shipment tracking and delivery info
ActivityLog	Log_ID	–	Audit trail for system operations


customers
id
first_name
last_name
phone
email
created_at
updated_at
addresses
id
address_line_1
address_line_2
city
country
postal_code
customer_id (FK)
products
id
name
description
sku
img_url
stock_qty
price
created_at
updated_at
sub_category_id (FK)
orders
id
currency
status
total
created_at
updated_at
customer_id (FK)
order_items
id
order_id (FK)
quantity
unit_price
discount
total
product_id (FK)
sub_categories
id
name
description
category_id (FK)
shipments
id
pck_number
tracking_number
carrier
status
created_at
updated_at
order_id (FK)
payments
id
method
status
amount
created_at
order_id (FK)
categories
id
name
description
activity_logs
id
table_name
record_id
operation
changed_data
changed_by
changed_at
ip_address
remarks

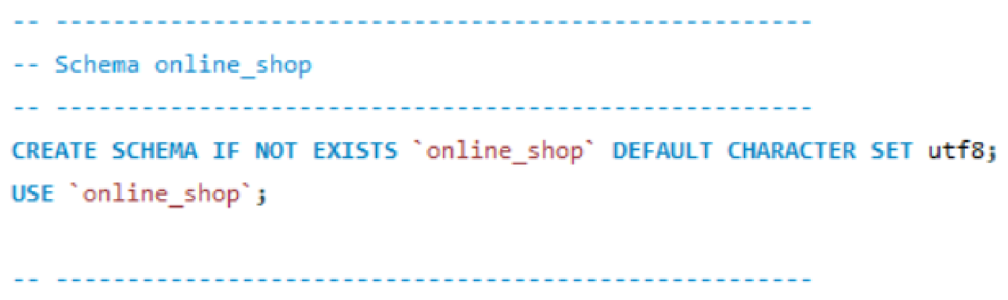
Constraints

- All primary keys are defined as INT AUTO_INCREMENT.
- Foreign keys use ON UPDATE CASCADE and ON DELETE CASCADE.
- Unique constraints on email, sku, and tracking_number.
- Check constraints applied to ensure valid status values for payments and orders.

SQL DDL (CREATE TABLE Statements)

The following section presents the SQL Data Definition Language (DDL) statements used to define the structure of the *Online Shop* database.
Each table was created in MySQL under the schema online_shop.
The DDL includes table creation commands, data types, primary and foreign keys, and integrity constraints.

Schema Creation



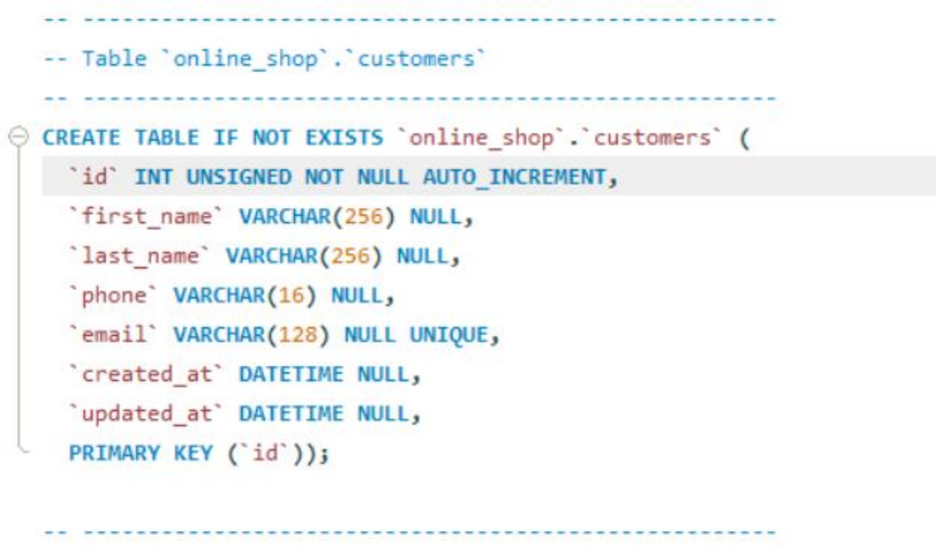
```
-- -----------------------------------------------------
-- Schema online_shop
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `online_shop` DEFAULT CHARACTER SET utf8;
USE `online_shop`;

-- -----------------------------------------------------
```

- This command initializes the database schema for the system.
  All tables were subsequently created inside this schema.

Table Definitions

Below are excerpts from the key table creation statements.
(Complete DDL script is available in the project GitHub repository.)

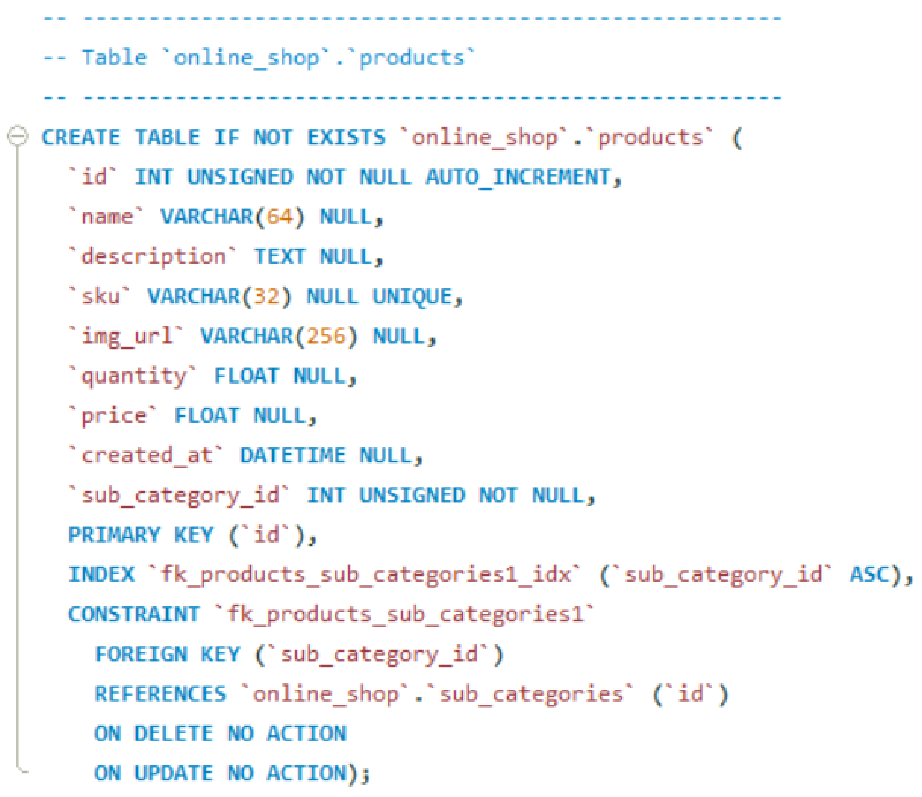
a) **Customers Table**



```
-- -----------------------------------------------------
-- Table `online_shop`.`customers`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `online_shop`.`customers` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `first_name` VARCHAR(256) NULL,
  `last_name` VARCHAR(256) NULL,
  `phone` VARCHAR(16) NULL,
  `email` VARCHAR(128) NULL UNIQUE,
  `created_at` DATETIME NULL,
  `updated_at` DATETIME NULL,
  PRIMARY KEY (`id`));

-- -----------------------------------------------------
```

- Stores customer information and enforces a unique email constraint

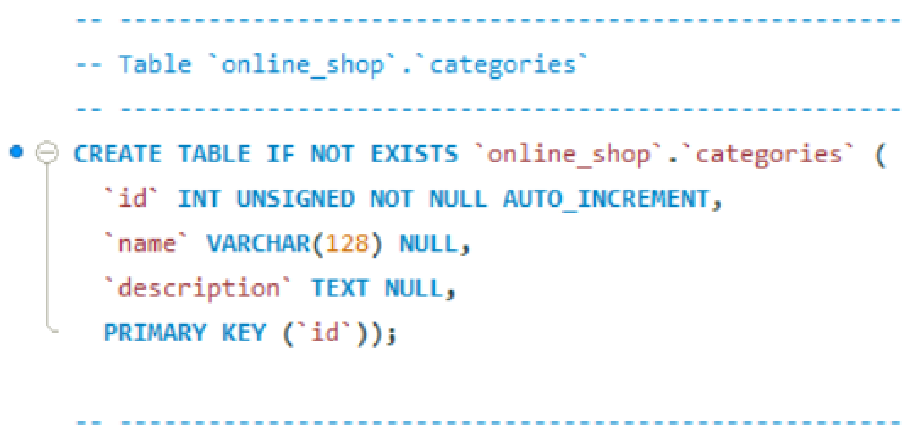
b) **Products Table**



```
-- -----------------------------------------------------
-- Table `online_shop`.`products`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `online_shop`.`products` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(64) NULL,
  `description` TEXT NULL,
  `sku` VARCHAR(32) NULL UNIQUE,
  `img_url` VARCHAR(256) NULL,
  `quantity` FLOAT NULL,
  `price` FLOAT NULL,
  `created_at` DATETIME NULL,
  `sub_category_id` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_products_sub_categories1_idx` (`sub_category_id` ASC),
  CONSTRAINT `fk_products_sub_categories1`
    FOREIGN KEY (`sub_category_id`)
    REFERENCES `online_shop`.`sub_categories` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION);
```

- Defines all products in the system and links them to their corresponding subcategory.

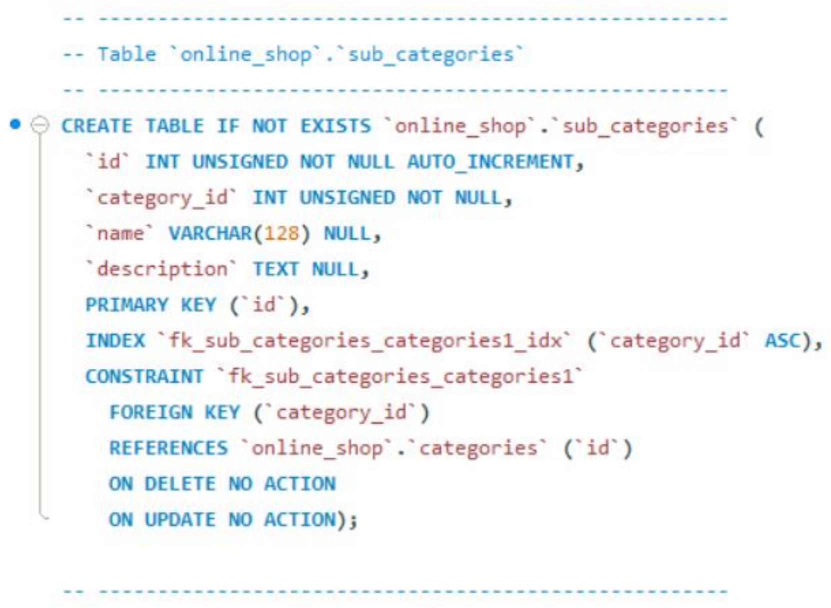
c) Categories Table



```
-- -----------------------------------------------------
-- Table `online_shop`.`categories`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `online_shop`.`categories` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(128) NULL,
  `description` TEXT NULL,
  PRIMARY KEY (`id`));

-- -----------------------------------------------------
```

- It contains high-level product categories (e.g., Electronics, Fashion). Categories provide the top-level classification used to group related subcategories and products.

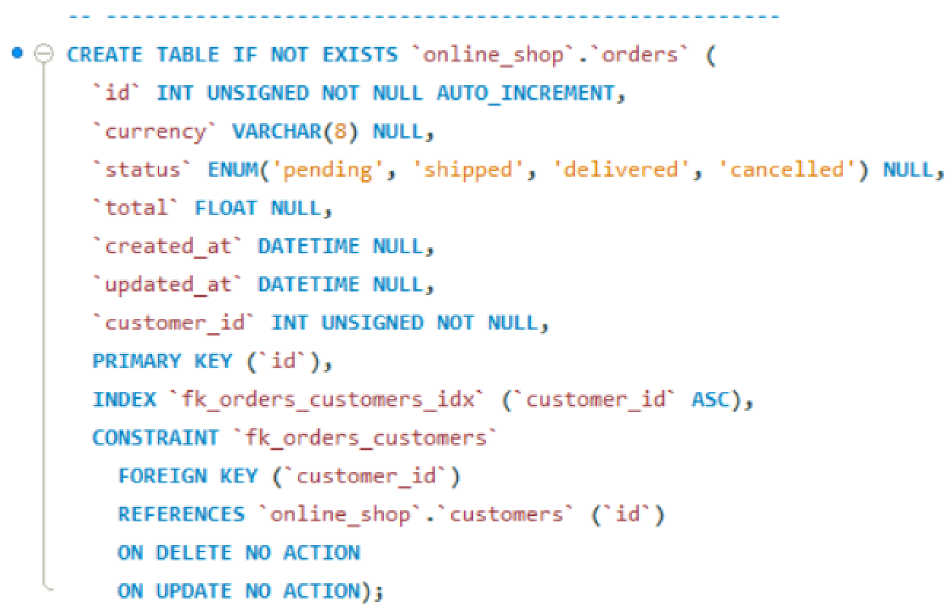
d) Sub_Categories Table



```
-- -----------------------------------------------------
-- Table `online_shop`.`sub_categories`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `online_shop`.`sub_categories` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `category_id` INT UNSIGNED NOT NULL,
  `name` VARCHAR(128) NULL,
  `description` TEXT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_sub_categories_categories1_idx` (`category_id` ASC),
  CONSTRAINT `fk_sub_categories_categories1`
    FOREIGN KEY (`category_id`)
    REFERENCES `online_shop`.`categories` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION);

-- -----------------------------------------------------
```

- Represents subdivisions of product categories (e.g., Mobile Phones under Electronics). Each sub_categories record references the parent categories table through category_id, enforcing the category → subcategory relationship.

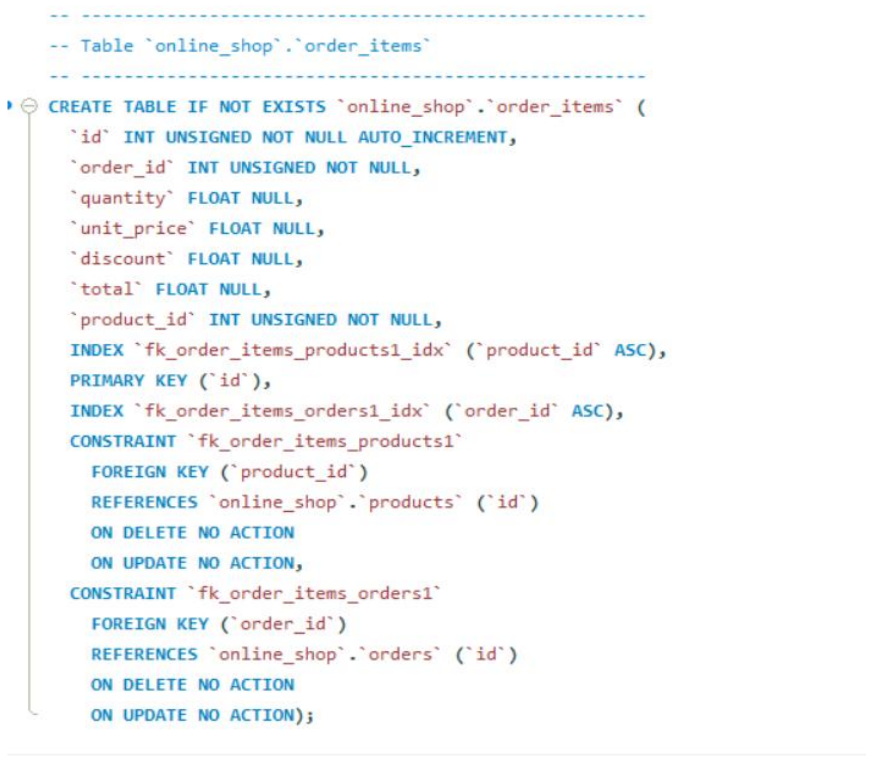
e) Orders Table



```
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `online_shop`.`orders` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `currency` VARCHAR(8) NULL,
  `status` ENUM('pending', 'shipped', 'delivered', 'cancelled') NULL,
  `total` FLOAT NULL,
  `created_at` DATETIME NULL,
  `updated_at` DATETIME NULL,
  `customer_id` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_orders_customers_idx` (`customer_id` ASC),
  CONSTRAINT `fk_orders_customers`
    FOREIGN KEY (`customer_id`)
    REFERENCES `online_shop`.`customers` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION);
```

- Represents customer purchase orders with order status and total cost.

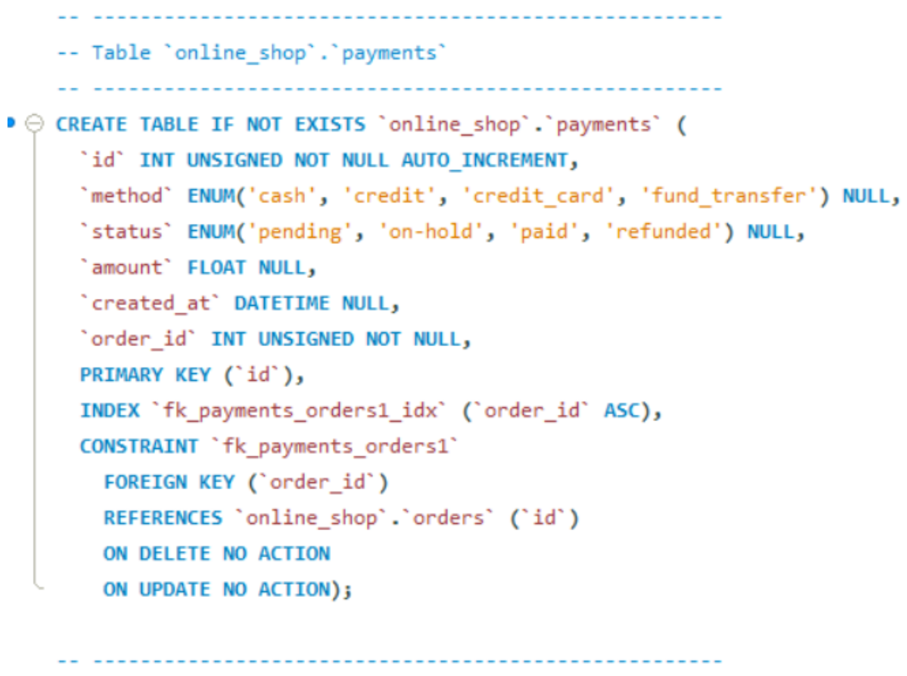
f) Order Items Table



```
-- -----------------------------------------------------
-- Table `online_shop`.`order_items`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `online_shop`.`order_items` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `order_id` INT UNSIGNED NOT NULL,
  `quantity` FLOAT NULL,
  `unit_price` FLOAT NULL,
  `discount` FLOAT NULL,
  `total` FLOAT NULL,
  `product_id` INT UNSIGNED NOT NULL,
  INDEX `fk_order_items_products1_idx` (`product_id` ASC),
  PRIMARY KEY (`id`),
  INDEX `fk_order_items_orders1_idx` (`order_id` ASC),
  CONSTRAINT `fk_order_items_products1`
    FOREIGN KEY (`product_id`)
    REFERENCES `online_shop`.`products` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_order_items_orders1`
    FOREIGN KEY (`order_id`)
    REFERENCES `online_shop`.`orders` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION);
```

- Contains the items included in each order and links products to orders.

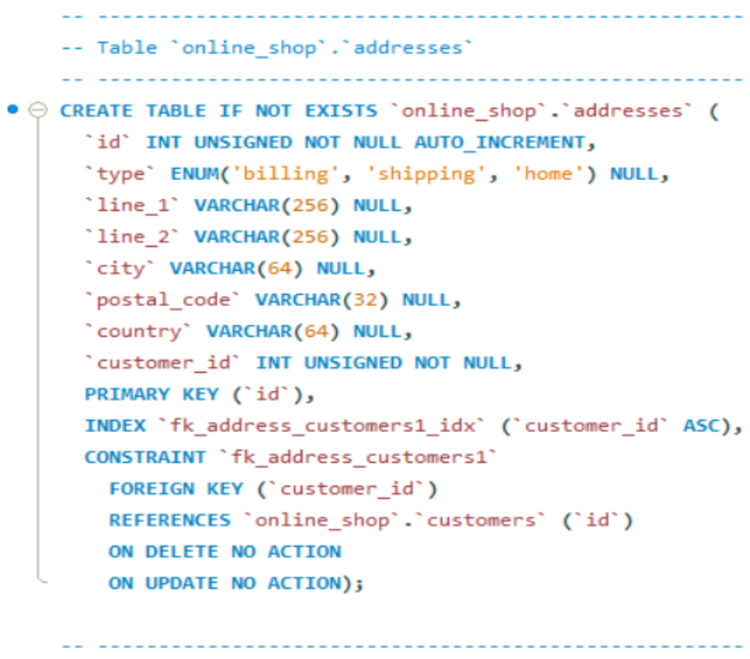
g) **Payments Table**



```
-- -----------------------------------------------------
-- Table `online_shop`.`payments`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `online_shop`.`payments` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `method` ENUM('cash', 'credit', 'credit_card', 'fund_transfer') NULL,
  `status` ENUM('pending', 'on-hold', 'paid', 'refunded') NULL,
  `amount` FLOAT NULL,
  `created_at` DATETIME NULL,
  `order_id` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_payments_orders1_idx` (`order_id` ASC),
  CONSTRAINT `fk_payments_orders1`
    FOREIGN KEY (`order_id`)
    REFERENCES `online_shop`.`orders` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION);

-- -----------------------------------------------------
```

- Records payment method, amount, and transaction status for each order.

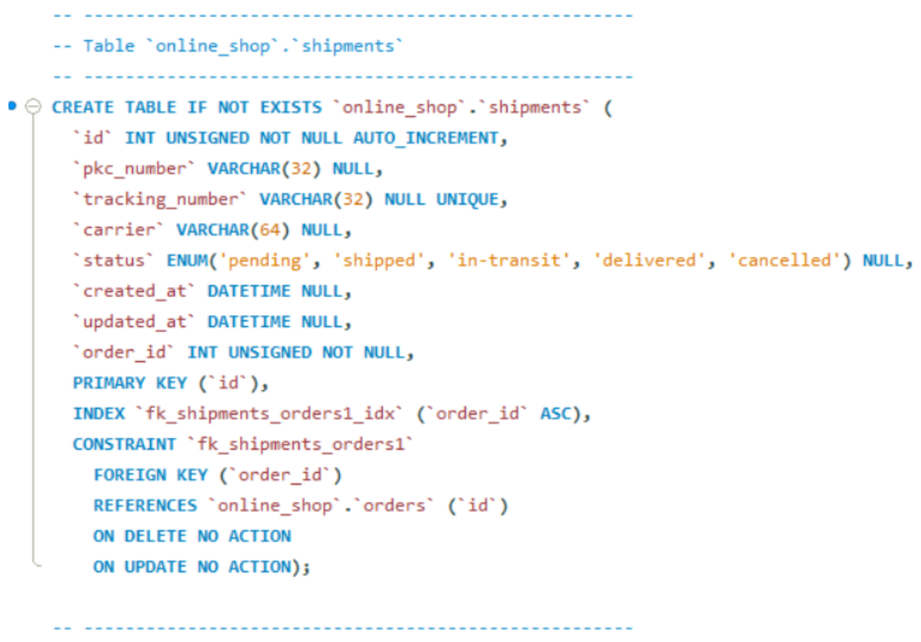
h) **Addresses Table**



```
-- -----------------------------------------------------
-- Table `online_shop`.`addresses`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `online_shop`.`addresses` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `type` ENUM('billing', 'shipping', 'home') NULL,
  `line_1` VARCHAR(256) NULL,
  `line_2` VARCHAR(256) NULL,
  `city` VARCHAR(64) NULL,
  `postal_code` VARCHAR(32) NULL,
  `country` VARCHAR(64) NULL,
  `customer_id` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_address_customers1_idx` (`customer_id` ASC),
  CONSTRAINT `fk_address_customers1`
    FOREIGN KEY (`customer_id`)
    REFERENCES `online_shop`.`customers` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION);

-- -----------------------------------------------------
```

- Stores customer addresses for billing and shipping. The type ENUM restricts entries to billing, shipping, or home. Each address links to a customers record via customer_id, allowing a customer to have multiple address types.

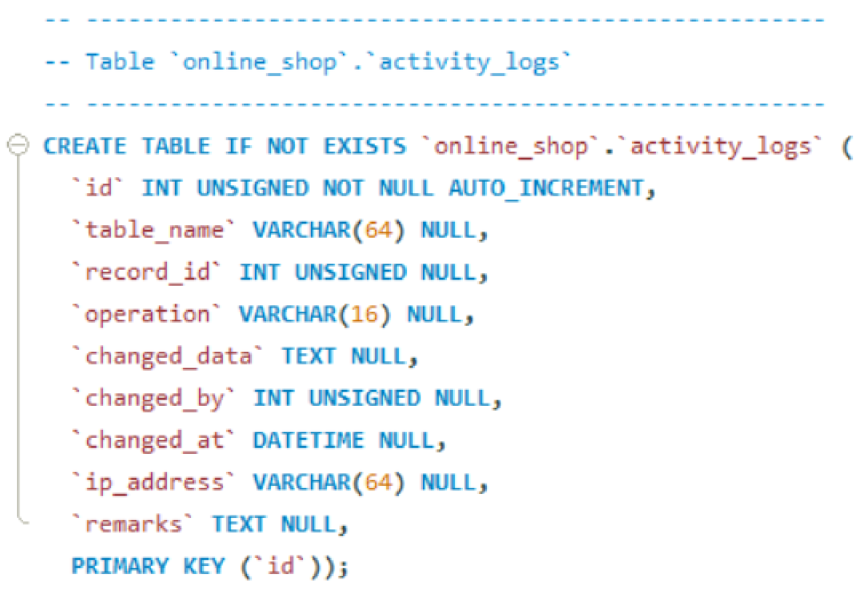
i) Shipments Table



```
-- -----------------------------------------------------
-- Table `online_shop`.`shipments`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `online_shop`.`shipments` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `pkc_number` VARCHAR(32) NULL,
  `tracking_number` VARCHAR(32) NULL UNIQUE,
  `carrier` VARCHAR(64) NULL,
  `status` ENUM('pending', 'shipped', 'in-transit', 'delivered', 'cancelled') NULL,
  `created_at` DATETIME NULL,
  `updated_at` DATETIME NULL,
  `order_id` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_shipments_orders1_idx` (`order_id` ASC),
  CONSTRAINT `fk_shipments_orders1`
    FOREIGN KEY (`order_id`)
    REFERENCES `online_shop`.`orders` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION);

-- -----------------------------------------------------
```

- Tracks delivery information and shipment status for customer orders.

j) Activity Logs Table



```sql
-- -----------------------------------------------------
-- Table `online_shop`.`activity_logs`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `online_shop`.`activity_logs` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `table_name` VARCHAR(64) NULL,
  `record_id` INT UNSIGNED NULL,
  `operation` VARCHAR(16) NULL,
  `changed_data` TEXT NULL,
  `changed_by` INT UNSIGNED NULL,
  `changed_at` DATETIME NULL,
  `ip_address` VARCHAR(64) NULL,
  `remarks` TEXT NULL,
  PRIMARY KEY (`id`));
```

- Provides a system-wide audit log of all data modification operations.

SQL DML (INSERT sample data)

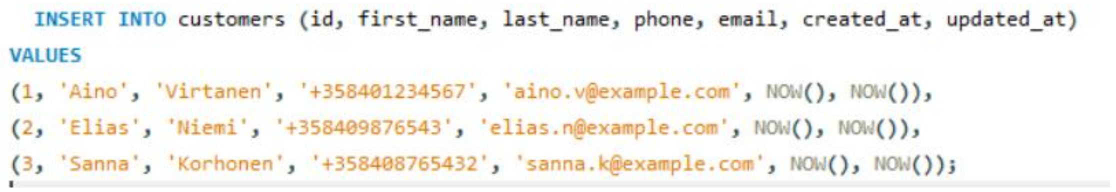
Purpose

The SQL DML (Data Manipulation Language) script populates the **Online Shop database** with realistic sample data for testing and demonstration.
It includes INSERT statements to create initial records, as well as UPDATE and DELETE operations to simulate real business processes such as modifying customer details, updating product stock, and archiving old orders.

Sample Data Insertions

Below are representative examples of how sample data was inserted into the core tables. (The complete DML script is available in the GitHub repository.)

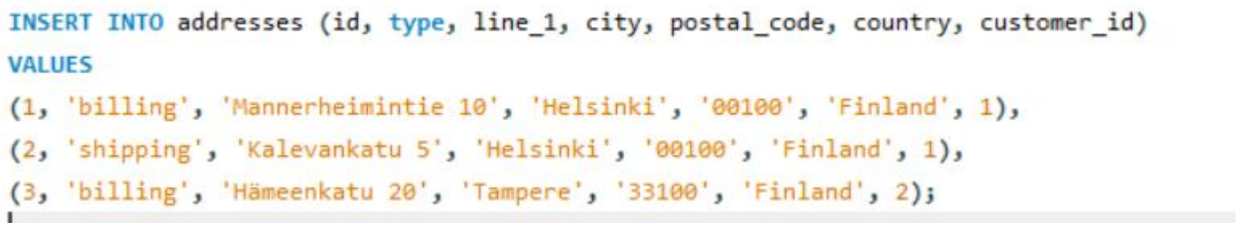
a) Customers



```sql
 INSERT INTO customers (id, first_name, last_name, phone, email, created_at, updated_at)
VALUES
(1, 'Aino', 'Virtanen', '+358401234567', 'aino.v@example.com', NOW(), NOW()),
(2, 'Elias', 'Niemi', '+358409876543', 'elias.n@example.com', NOW(), NOW()),
(3, 'Sanna', 'Korhonen', '+358408765432', 'sanna.k@example.com', NOW(), NOW());
```

- Adds three sample customers with realistic Finnish names, emails, and timestamps to test relationships with orders and addresses.

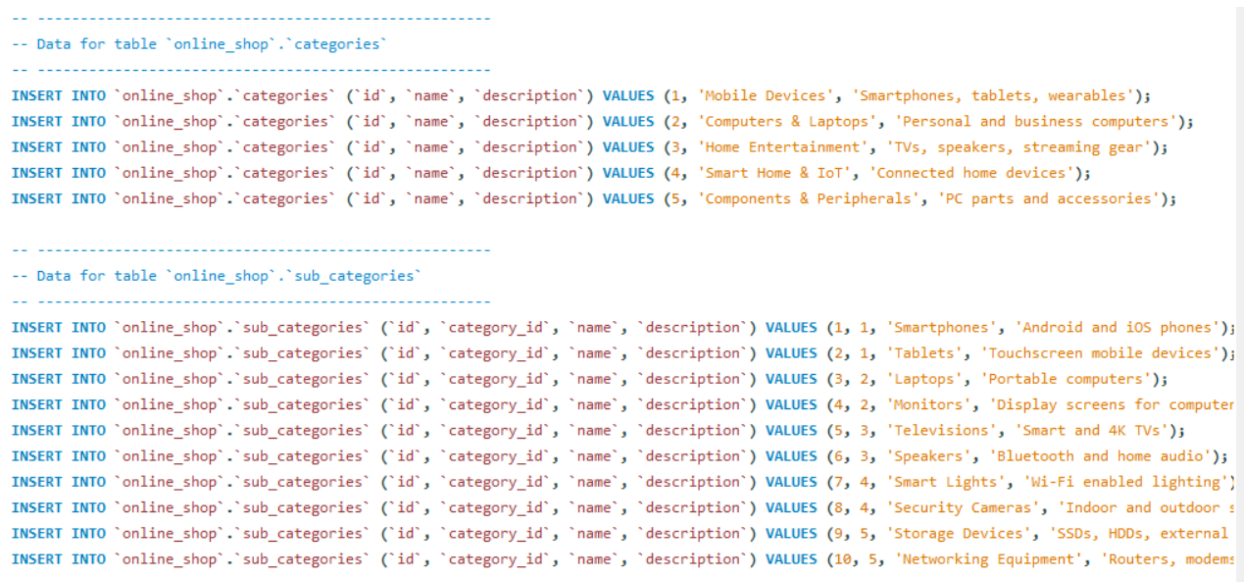
b) Addresses



```sql
INSERT INTO addresses (id, type, line_1, city, postal_code, country, customer_id)
VALUES
(1, 'billing', 'Mannerheimintie 10', 'Helsinki', '00100', 'Finland', 1),
(2, 'shipping', 'Kalevankatu 5', 'Helsinki', '00100', 'Finland', 1),
(3, 'billing', 'Hämeenkatu 20', 'Tampere', '33100', 'Finland', 2);
```

- Links each address record to its customer using customer_id, showing one-to-many relationships.

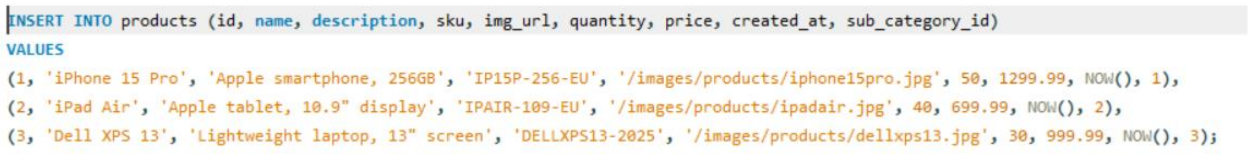
c) Categories and Subcategories



```
-- ----------------------------------------------------
-- Data for table `online_shop`.`categories`
-- ----------------------------------------------------
INSERT INTO `online_shop`.`categories` (`id`, `name`, `description`) VALUES (1, 'Mobile Devices', 'Smartphones, tablets, wearables');
INSERT INTO `online_shop`.`categories` (`id`, `name`, `description`) VALUES (2, 'Computers & Laptops', 'Personal and business computers');
INSERT INTO `online_shop`.`categories` (`id`, `name`, `description`) VALUES (3, 'Home Entertainment', 'TVs, speakers, streaming gear');
INSERT INTO `online_shop`.`categories` (`id`, `name`, `description`) VALUES (4, 'Smart Home & IoT', 'Connected home devices');
INSERT INTO `online_shop`.`categories` (`id`, `name`, `description`) VALUES (5, 'Components & Peripherals', 'PC parts and accessories');

-- ----------------------------------------------------
-- Data for table `online_shop`.`sub_categories`
-- ----------------------------------------------------
INSERT INTO `online_shop`.`sub_categories` (`id`, `category_id`, `name`, `description`) VALUES (1, 1, 'Smartphones', 'Android and iOS phones');
INSERT INTO `online_shop`.`sub_categories` (`id`, `category_id`, `name`, `description`) VALUES (2, 1, 'Tablets', 'Touchscreen mobile devices');
INSERT INTO `online_shop`.`sub_categories` (`id`, `category_id`, `name`, `description`) VALUES (3, 2, 'Laptops', 'Portable computers');
INSERT INTO `online_shop`.`sub_categories` (`id`, `category_id`, `name`, `description`) VALUES (4, 2, 'Monitors', 'Display screens for computer
INSERT INTO `online_shop`.`sub_categories` (`id`, `category_id`, `name`, `description`) VALUES (5, 3, 'Televisions', 'Smart and 4K TVs');
INSERT INTO `online_shop`.`sub_categories` (`id`, `category_id`, `name`, `description`) VALUES (6, 3, 'Speakers', 'Bluetooth and home audio');
INSERT INTO `online_shop`.`sub_categories` (`id`, `category_id`, `name`, `description`) VALUES (7, 4, 'Smart Lights', 'Wi-Fi enabled lighting')
INSERT INTO `online_shop`.`sub_categories` (`id`, `category_id`, `name`, `description`) VALUES (8, 4, 'Security Cameras', 'Indoor and outdoor s
INSERT INTO `online_shop`.`sub_categories` (`id`, `category_id`, `name`, `description`) VALUES (9, 5, 'Storage Devices', 'SSDs, HDDs, external
INSERT INTO `online_shop`.`sub_categories` (`id`, `category_id`, `name`, `description`) VALUES (10, 5, 'Networking Equipment', 'Routers, modems
```

- Establishes hierarchical product classification for the catalog.

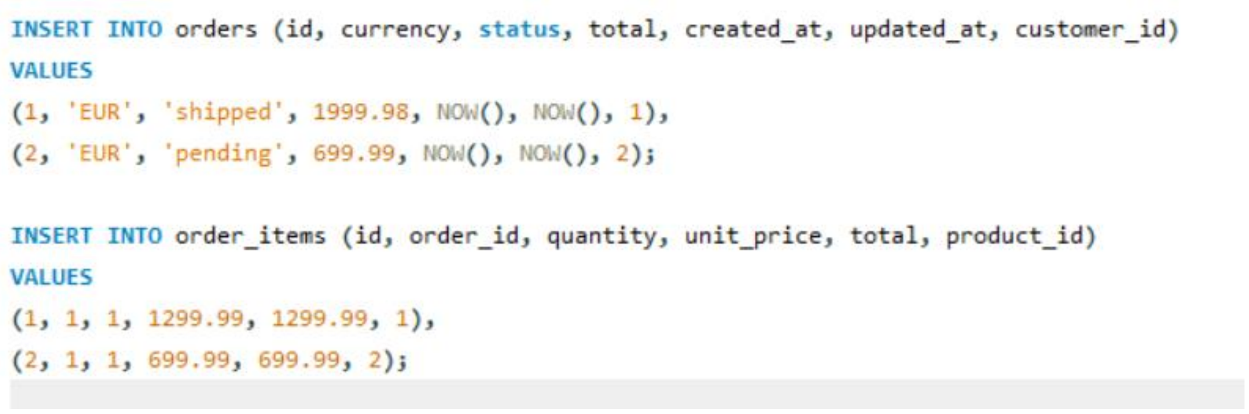
d) Products



```
INSERT INTO products (id, name, description, sku, img_url, quantity, price, created_at, sub_category_id)
VALUES
(1, 'iPhone 15 Pro', 'Apple smartphone, 256GB', 'IP15P-256-EU', '/images/products/iphone15pro.jpg', 50, 1299.99, NOW(), 1),
(2, 'iPad Air', 'Apple tablet, 10.9" display', 'IPAIR-109-EU', '/images/products/ipadair.jpg', 40, 699.99, NOW(), 2),
(3, 'Dell XPS 13', 'Lightweight laptop, 13" screen', 'DELLXPS13-2025', '/images/products/dellxps13.jpg', 30, 999.99, NOW(), 3);
```

- Populates the products table with sample items representing different subcategories (smartphones, tablets, laptops).
- Each record includes unique identifiers such as sku and an associated img_url for product images.
- The sub_category_id column enforces referential integrity by linking each product to its respective subcategory in the sub_categories table.

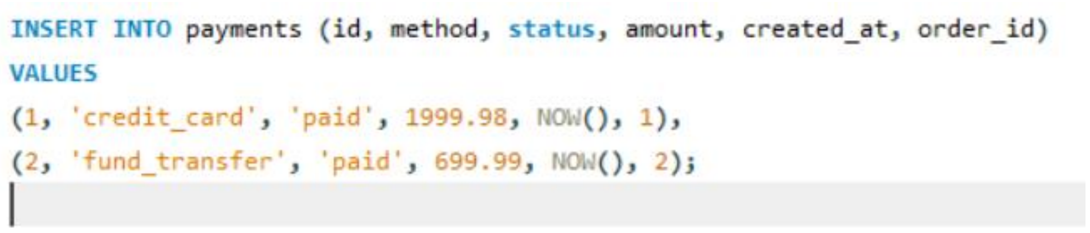
e) Orders and Order Items



```
INSERT INTO orders (id, currency, status, total, created_at, updated_at, customer_id)
VALUES
(1, 'EUR', 'shipped', 1999.98, NOW(), NOW(), 1),
(2, 'EUR', 'pending', 699.99, NOW(), NOW(), 2);

INSERT INTO order_items (id, order_id, quantity, unit_price, total, product_id)
VALUES
(1, 1, 1, 1299.99, 1299.99, 1),
(2, 1, 1, 699.99, 699.99, 2);
```

- Creates relationships between customers and products through orders and order items.

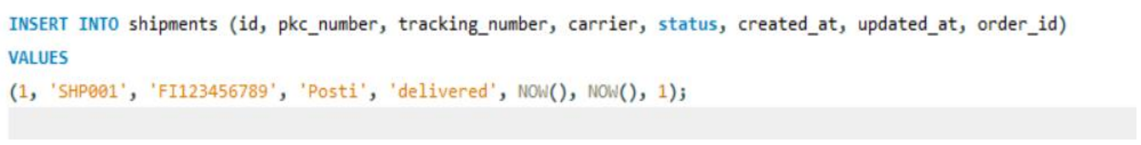
f) Payments



```
INSERT INTO payments (id, method, status, amount, created_at, order_id)
VALUES
(1, 'credit_card', 'paid', 1999.98, NOW(), 1),
(2, 'fund_transfer', 'paid', 699.99, NOW(), 2);
```

- Stores order payment information linked to the orders table by order_id.

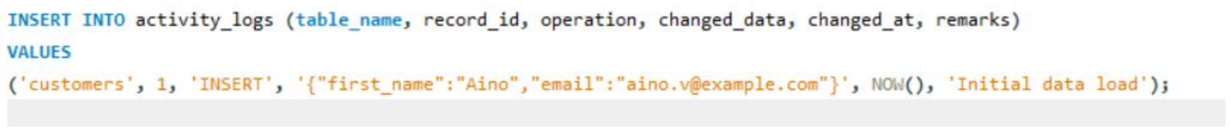
g) Shipments



```
INSERT INTO shipments (id, pkc_number, tracking_number, carrier, status, created_at, updated_at, order_id)
VALUES
(1, 'SHP001', 'FI123456789', 'Posti', 'delivered', NOW(), NOW(), 1);
```

- Simulates logistics operations using sample tracking numbers and delivery statuses.

h) Activity Logs



```sql
INSERT INTO activity_logs (table_name, record_id, operation, changed_data, changed_at, remarks)
VALUES
('customers', 1, 'INSERT', '{"first_name":"Aino","email":"aino.v@example.com"}', NOW(), 'Initial data load');
```

- Maintains an audit trail for every insert operation performed on core tables.

Data Modification and Deletion Operations

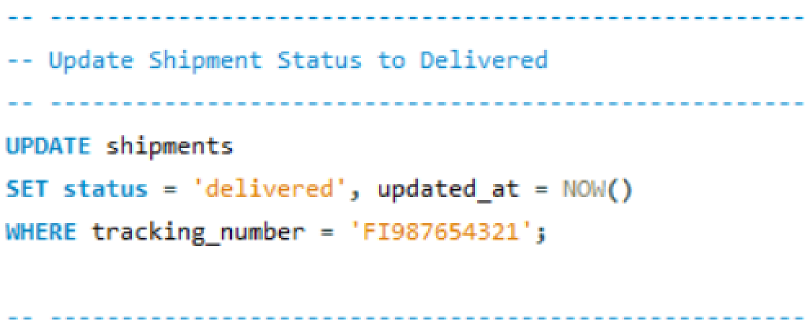
The DML script also includes advanced SQL operations to maintain data integrity and simulate real-world updates:

- For example,

```sql
-- ------------------------------------------------------
-- Update Customer Email Format
-- ------------------------------------------------------
UPDATE customers
SET email = LOWER(email)
WHERE email IS NOT NULL;

-- ------------------------------------------------------
```

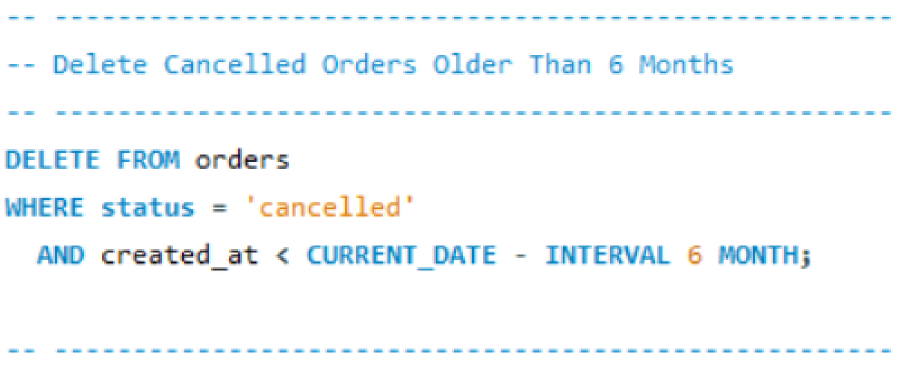
was used to ensure consistent email formatting across all records.

- Shipment statuses were updated to reflect delivery progress using statements such as,



```
-- ---------------------------------------------------
-- Update Shipment Status to Delivered
-- ---------------------------------------------------
UPDATE shipments
SET status = 'delivered', updated_at = NOW()
WHERE tracking_number = 'FI987654321';

-- ---------------------------------------------------
```

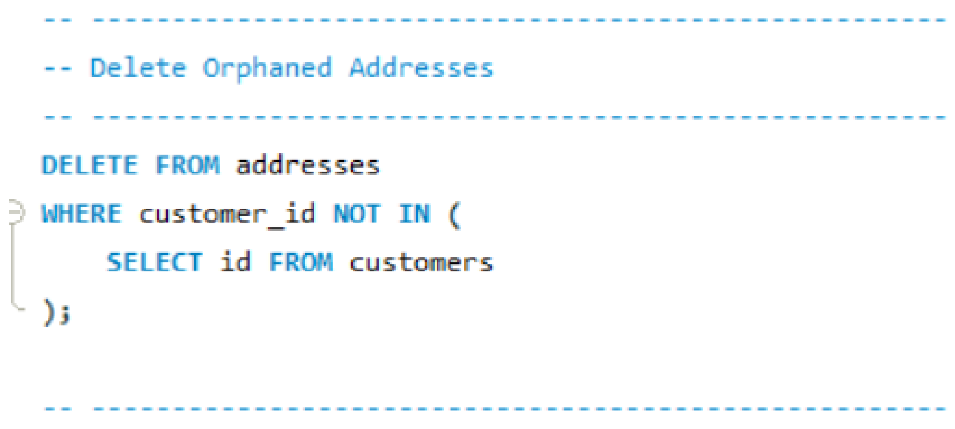
- To remove outdated or unnecessary records, the following deletions were applied:



```
-- ---------------------------------------------------
-- Delete Cancelled Orders Older Than 6 Months
-- ---------------------------------------------------
DELETE FROM orders
WHERE status = 'cancelled'
  AND created_at < CURRENT_DATE - INTERVAL 6 MONTH;

-- ---------------------------------------------------
```

which cleans up cancelled orders older than six months.

- Moreover,



```
-- ----------------------------------------------------
-- Delete Orphaned Addresses
-- ----------------------------------------------------
DELETE FROM addresses
WHERE customer_id NOT IN (
    SELECT id FROM customers
);

-- ----------------------------------------------------
```

which removes orphaned address records left behind after customer deletions. These operations help preserve database consistency, accuracy, and performance.

Summary of DML Implementation

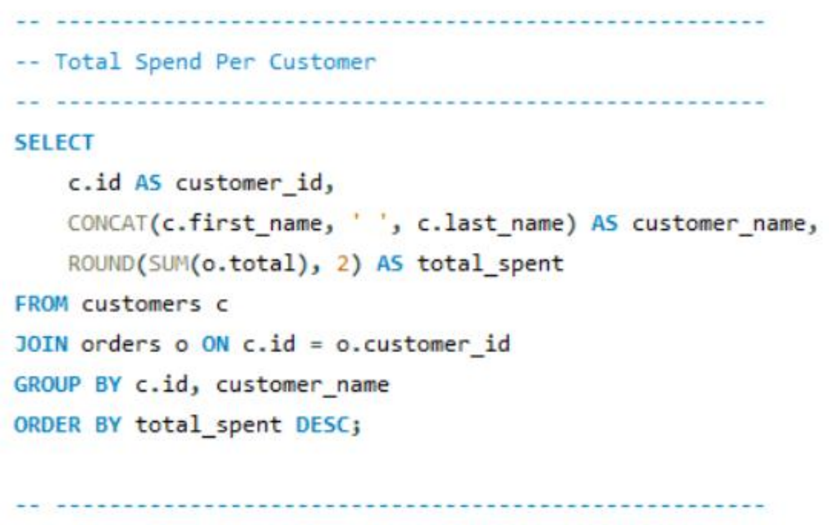
The sample data ensures full referential integrity across all entities:

- Customers link to addresses and orders.
- Orders link to payments, shipments, and order items.
- Products are connected to subcategories and categories.
- Activity logs track data changes for auditing purposes.
- The dataset enables meaningful SQL queries, aggregates, and subqueries, providing a realistic foundation for testing the online shop database.

SQL Queries (Joins, Aggregates, and Subqueries)

- This section demonstrates meaningful SQL SELECT queries that test and validate the **Online Shop database.**
  The queries use joins, aggregates, and subqueries to extract business insights and verify the correctness of relationships among entities such as customers, orders, products, payments, and shipments.
  All queries were executed in MySQL Workbench using the sample data from the DML script.

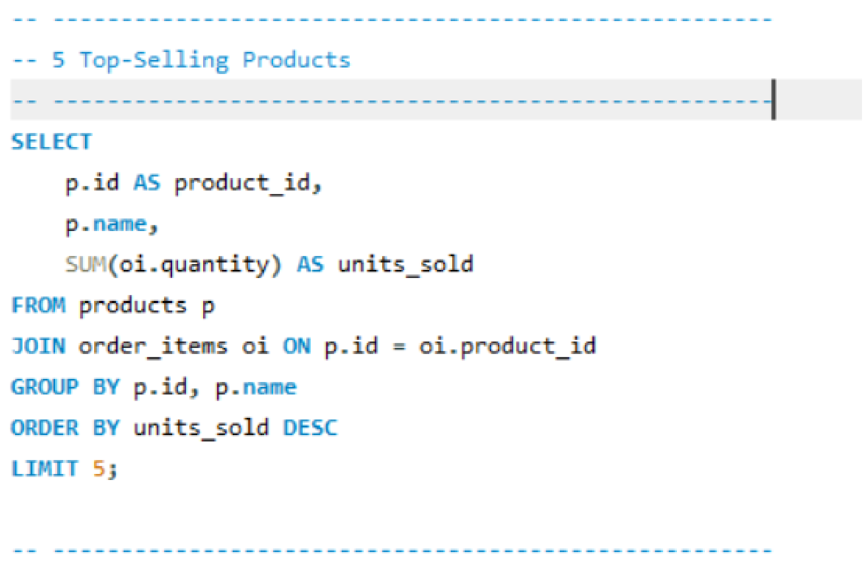
a) Total Spend per Customer



```
-- -----------------------------------------------------
-- Total Spend Per Customer
-- -----------------------------------------------------
SELECT
    c.id AS customer_id,
    CONCAT(c.first_name, ' ', c.last_name) AS customer_name,
    ROUND(SUM(o.total), 2) AS total_spent
FROM customers c
JOIN orders o ON c.id = o.customer_id
GROUP BY c.id, customer_name
ORDER BY total_spent DESC;

-- -----------------------------------------------------
```

- Calculates the total amount each customer has spent by joining customers and orders.
  Uses aggregation (SUM) and grouping (GROUP BY) to produce per-customer totals.

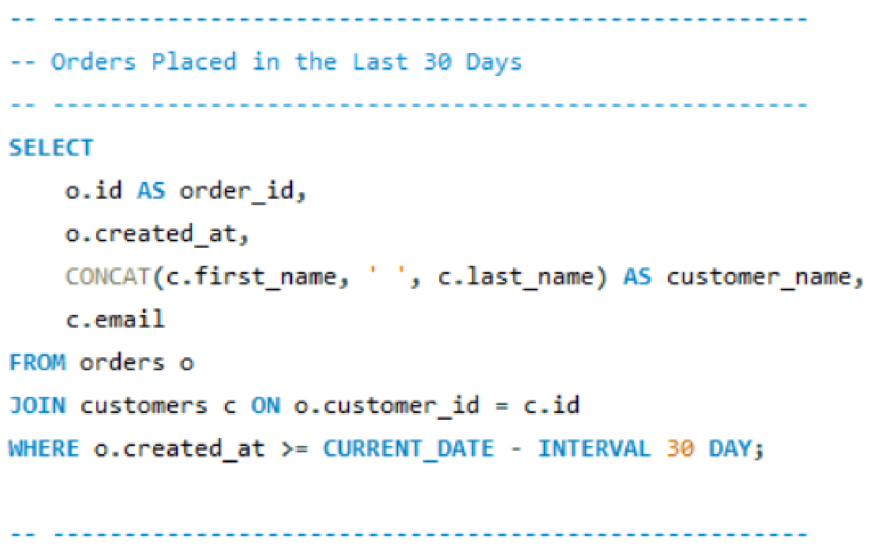
b) Top 5 Best-Selling Products



```sql
-- ----------------------------------------------------
-- 5 Top-Selling Products
-- ----------------------------------------------------
SELECT
    p.id AS product_id,
    p.name,
    SUM(oi.quantity) AS units_sold
FROM products p
JOIN order_items oi ON p.id = oi.product_id
GROUP BY p.id, p.name
ORDER BY units_sold DESC
LIMIT 5;

-- ----------------------------------------------------
```

- Finds the five most-sold products. Uses aggregation (SUM) with ordering and limiting for reporting top performers.

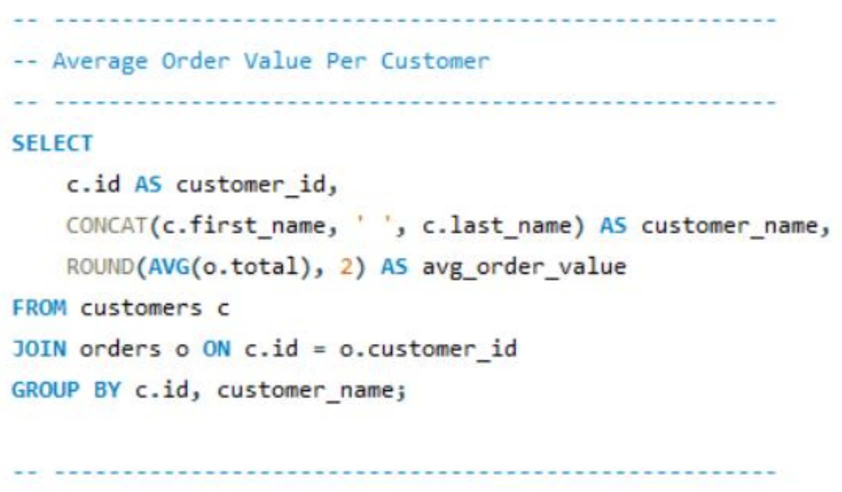
c) Orders Placed in the Last 30 Days



```sql
-- ----------------------------------------------------
-- Orders Placed in the Last 30 Days
-- ----------------------------------------------------
SELECT
    o.id AS order_id,
    o.created_at,
    CONCAT(c.first_name, ' ', c.last_name) AS customer_name,
    c.email
FROM orders o
JOIN customers c ON o.customer_id = c.id
WHERE o.created_at >= CURRENT_DATE - INTERVAL 30 DAY;

-- ----------------------------------------------------
```

- Retrieves all recent orders within the last month, showing how date filtering and joins can identify active customers.

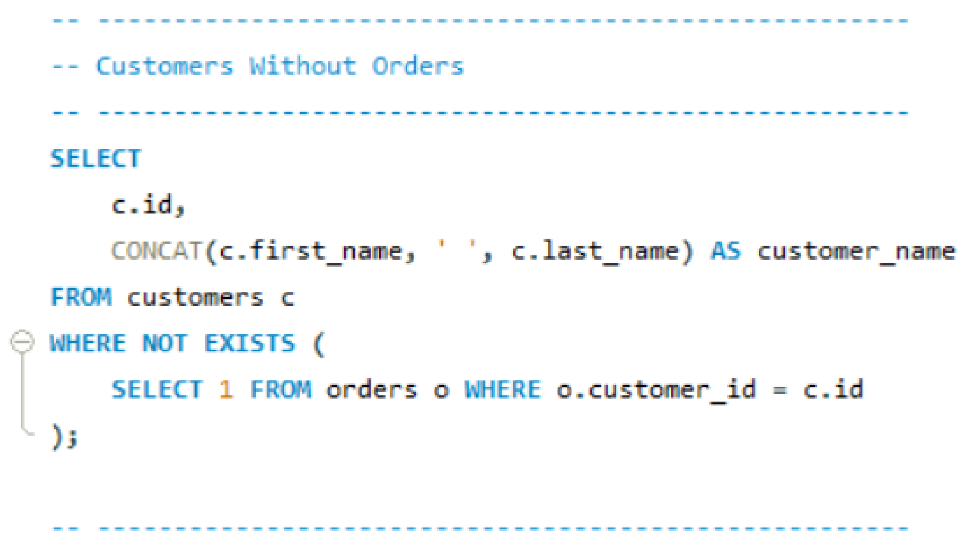
d) Average Order Value per Customer



```sql
-- ---------------------------------------------------
-- Average Order Value Per Customer
-- ---------------------------------------------------
SELECT
    c.id AS customer_id,
    CONCAT(c.first_name, ' ', c.last_name) AS customer_name,
    ROUND(AVG(o.total), 2) AS avg_order_value
FROM customers c
JOIN orders o ON c.id = o.customer_id
GROUP BY c.id, customer_name;

-- ---------------------------------------------------
```

- Calculates the average order total per customer using the AVG function. Useful for analyzing customer purchasing behavior.

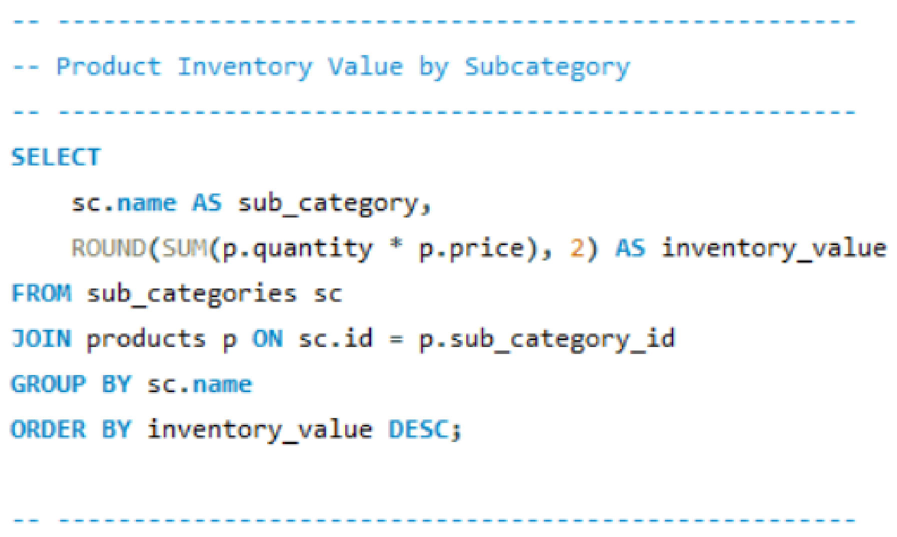
e) Customers Without Orders



```sql
-- ---------------------------------------------------
-- Customers Without Orders
-- ---------------------------------------------------
SELECT
    c.id,
    CONCAT(c.first_name, ' ', c.last_name) AS customer_name
FROM customers c
WHERE NOT EXISTS (
    SELECT 1 FROM orders o WHERE o.customer_id = c.id
);

-- ---------------------------------------------------
```

- Uses a subquery to identify customers who haven't placed any orders. Demonstrates the use of the NOT EXISTS condition for dependency checks.

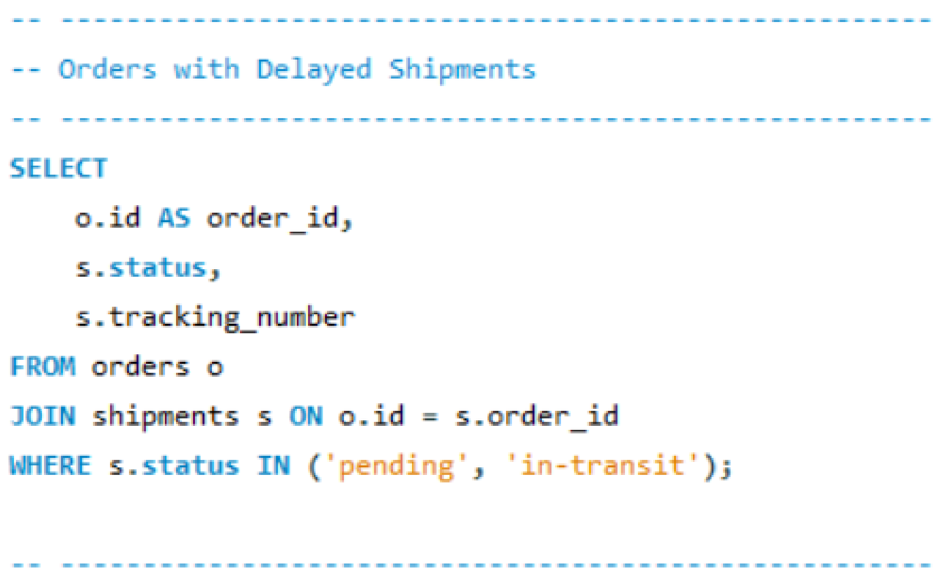
f) Product Inventory Value by Subcategory



```sql
-- -----------------------------------------------------
-- Product Inventory Value by Subcategory
-- -----------------------------------------------------
SELECT
    sc.name AS sub_category,
    ROUND(SUM(p.quantity * p.price), 2) AS inventory_value
FROM sub_categories sc
JOIN products p ON sc.id = p.sub_category_id
GROUP BY sc.name
ORDER BY inventory_value DESC;

-- -----------------------------------------------------
```

- Calculates the total inventory value per subcategory using arithmetic expressions and grouping.

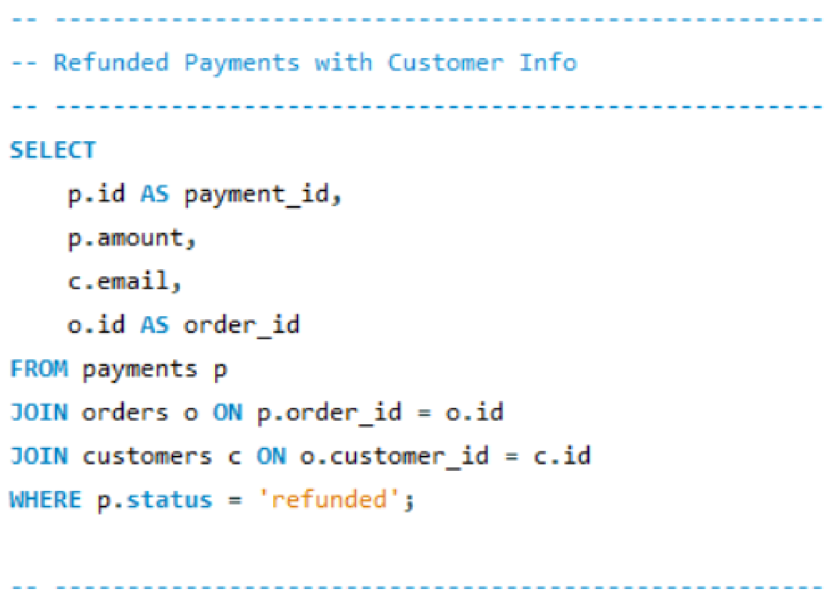
g) **Orders with Delayed Shipments**



```
-- ----------------------------------------------------
-- Orders with Delayed Shipments
-- ----------------------------------------------------
SELECT
    o.id AS order_id,
    s.status,
    s.tracking_number
FROM orders o
JOIN shipments s ON o.id = s.order_id
WHERE s.status IN ('pending', 'in-transit');

-- ----------------------------------------------------
```

- Identifies orders whose shipments are still pending or in transit. Demonstrates filtering on joined tables.

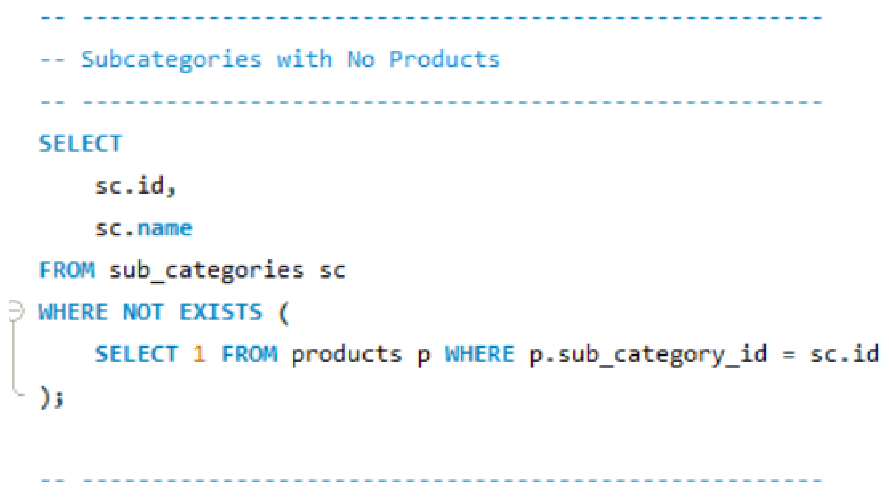
h) **Refunded Payments with Customer Info**



```
-- ----------------------------------------------------
-- Refunded Payments with Customer Info
-- ----------------------------------------------------
SELECT
    p.id AS payment_id,
    p.amount,
    c.email,
    o.id AS order_id
FROM payments p
JOIN orders o ON p.order_id = o.id
JOIN customers c ON o.customer_id = c.id
WHERE p.status = 'refunded';

-- ----------------------------------------------------
```

- Retrieves all refunded payments and their associated customer and order details. Combines multiple joins and a conditional filter.

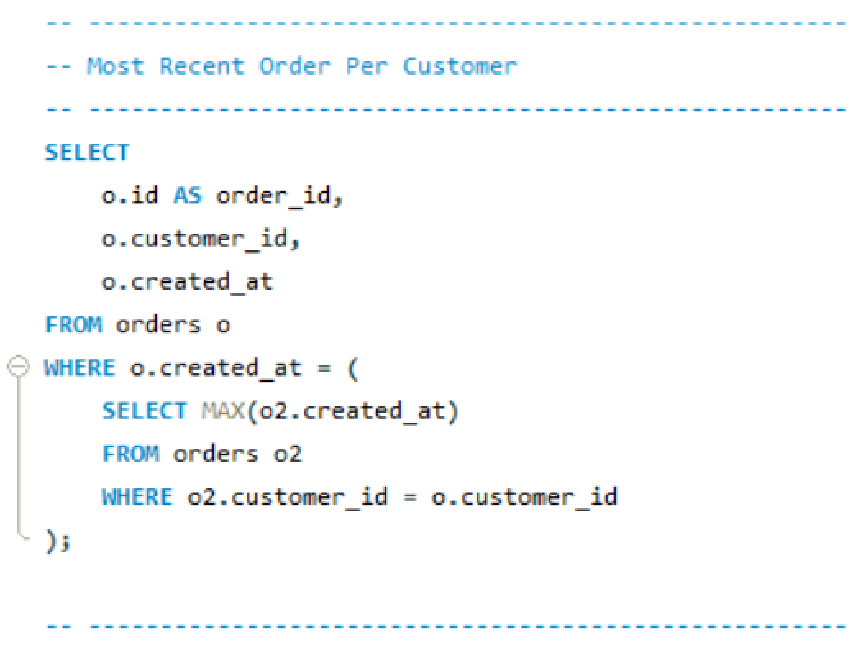
i) Subcategories Without Products



```sql
-- ---------------------------------------------------
-- Subcategories with No Products
-- ---------------------------------------------------
SELECT
    sc.id,
    sc.name
FROM sub_categories sc
WHERE NOT EXISTS (
    SELECT 1 FROM products p WHERE p.sub_category_id = sc.id
);

-- ---------------------------------------------------
```

- Uses a subquery to find empty subcategories.
  This helps in identifying categories that need new product additions.

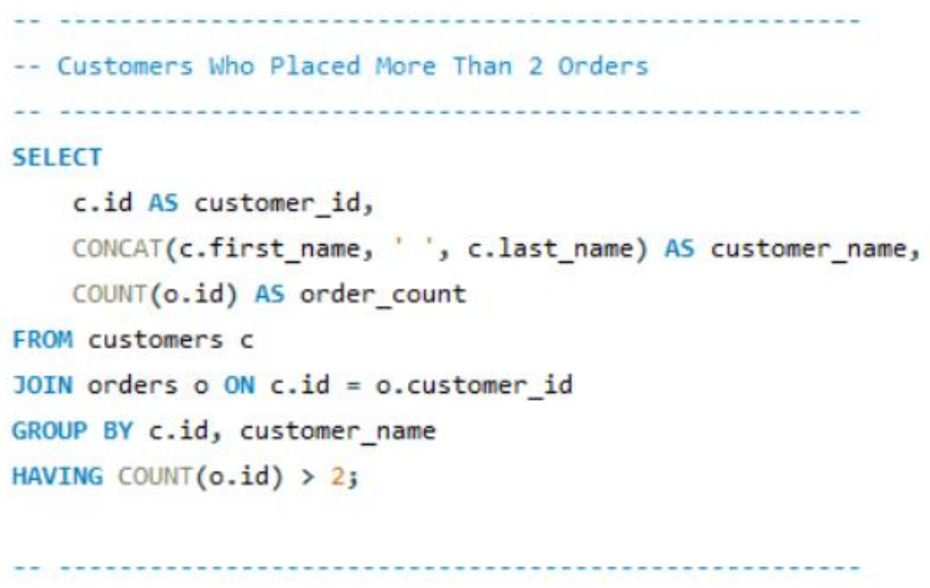
j) Most Recent Order per Customer



```sql
-- ----------------------------------------------------
-- Most Recent Order Per Customer
-- ----------------------------------------------------
SELECT
    o.id AS order_id,
    o.customer_id,
    o.created_at
FROM orders o
WHERE o.created_at = (
    SELECT MAX(o2.created_at)
    FROM orders o2
    WHERE o2.customer_id = o.customer_id
);

-- ----------------------------------------------------
```

- Shows each customer’s latest order using a correlated subquery and the MAX() function to find the most recent date.

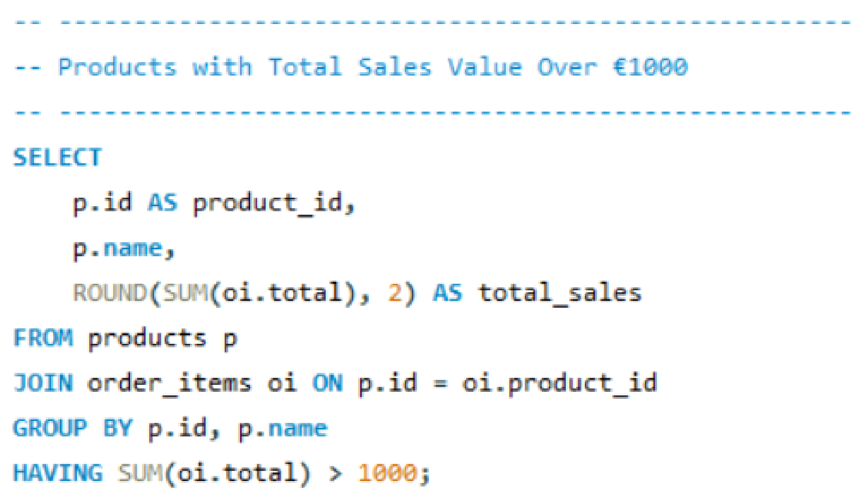
k) Customers with More Than Two Orders



```sql
-- ----------------------------------------------------
-- Customers Who Placed More Than 2 Orders
-- ----------------------------------------------------
SELECT
    c.id AS customer_id,
    CONCAT(c.first_name, ' ', c.last_name) AS customer_name,
    COUNT(o.id) AS order_count
FROM customers c
JOIN orders o ON c.id = o.customer_id
GROUP BY c.id, customer_name
HAVING COUNT(o.id) > 2;

-- ----------------------------------------------------
```

- Counts the number of orders per customer and filters to show only those with more than two orders using the HAVING clause.

l) Products with Total Sales Over €1000



```sql
-- ---------------------------------------------------
-- Products with Total Sales Value Over €1000
-- ---------------------------------------------------
SELECT
    p.id AS product_id,
    p.name,
    ROUND(SUM(oi.total), 2) AS total_sales
FROM products p
JOIN order_items oi ON p.id = oi.product_id
GROUP BY p.id, p.name
HAVING SUM(oi.total) > 1000;
```

Lists products with total sales exceeding €1000, showing aggregation and the use of a conditional HAVING filter for high-performing items.

Normalization Proof (1NF → 3NF)

First Normal Form (1NF)

Each column contains only atomic values.

No repeating groups or multivalued attributes.

Example: Each Customer has one email and one phone number per record.

Achieved: All tables are in 1NF.

Second Normal Form (2NF)

Database is in 1NF.

All non-key attributes are fully functionally dependent on the whole primary key.

Example: In Order_Items, quantity and price depend on the entire (order_id, product_id) relationship.

Achieved: No partial dependencies exist.

Third Normal Form (3NF)

Database is in 2NF.

No transitive dependencies between non-key attributes.

Example: In Customers, email does not depend on phone number or name—only on the key id.

Achieved: All non-key attributes depend solely on their primary keys.

Conclusion

This project demonstrates the complete process of designing and implementing a relational database system for an online shop.

It includes the ER diagram, relational schema, SQL DDL and DML scripts, ten meaningful queries, and normalization proof from 1NF to 3NF.

The final design ensures:

- ➢ High data integrity
- ➢ Efficient data access
- ➢ Logical organization and minimal redundancy
- ➢ This fulfills all course requirements for Database Design and Programming.

References and Attachments

- GitHub Repository: https://github.com/lionwalker/hamk-database-design-and-programming
- ER Diagram (ER_Diagram.png)
- Relational Schema (Relational_Schema.png)
- SQL Scripts (DDL.sql, DML.sql, DQL.sql - located in the docs folder of the GitHub repository)
- Final Presentation Slides (located in the docs folder of the GitHub repository)