Database Design I 1DL301

Project – Milestone 4

Group 7

Course Co-ordinator

George Fakas

Group Members

Anudeep Battu
battu.anudeep.3425@student.uu.se

Moutushi Sen
moutushi.sen.9419@student.uu.se

Shivaranjani Thiyagarajan
shivaranjani.thiyagarajan.9467@student.uu.se

Dona Harshani Kokila Wickramasinghe
dona-harshani-kokila.wickramasinghe.5617@student.uu.se

UPPSALA UNIVERSITET

Chapter 1: Entity-Relationship Model

1.1 Introduction

This chapter presents the design and structure of the database for AltOnline AB. It includes an Enhanced Entity-Relationship (EER) diagram to represent the relationships between various entities, a hierarchical structure of departments, and derived attributes for data analysis and navigation.

The hierarchy diagram illustrates the top-down structure of departments, starting from a special root department for storing welcome text. Following this, the detailed EER diagram provides a comprehensive representation of all entities, their attributes, relationships, and participation constraints.

The model captures the interactions between users, products, departments, reviews, and orders while ensuring data integrity and scalability. Derived attributes are included to optimize key functionalities such as real-time pricing calculations, breadcrumb generation for navigation, and dynamic product availability updates.

1.2 Hierarchy Diagram

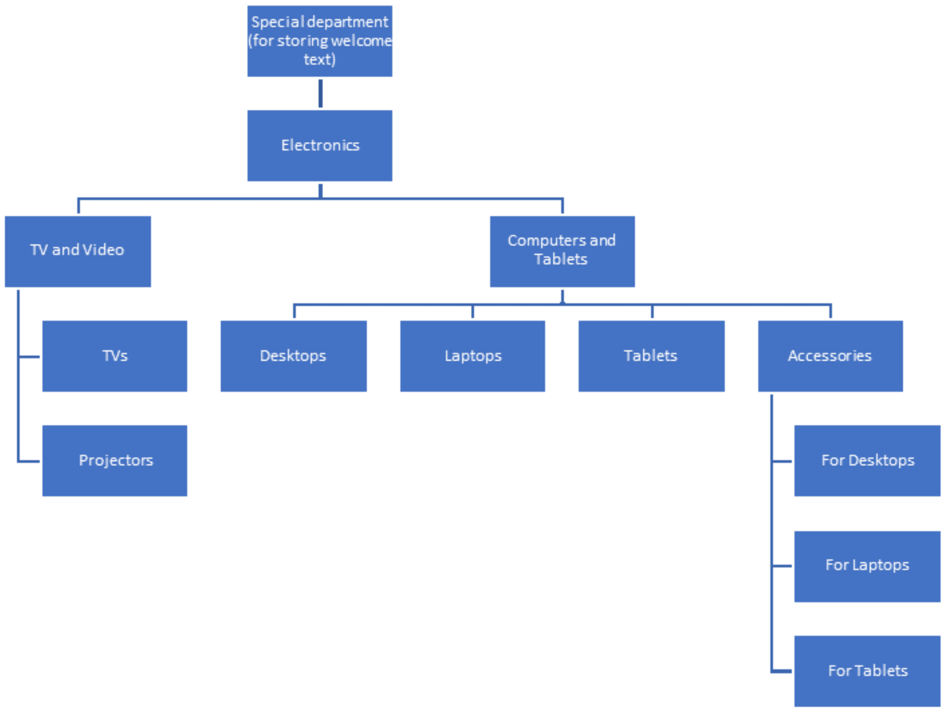


Figure 1.1: Hierarchy Diagram of Departments

1.3 EER Diagram

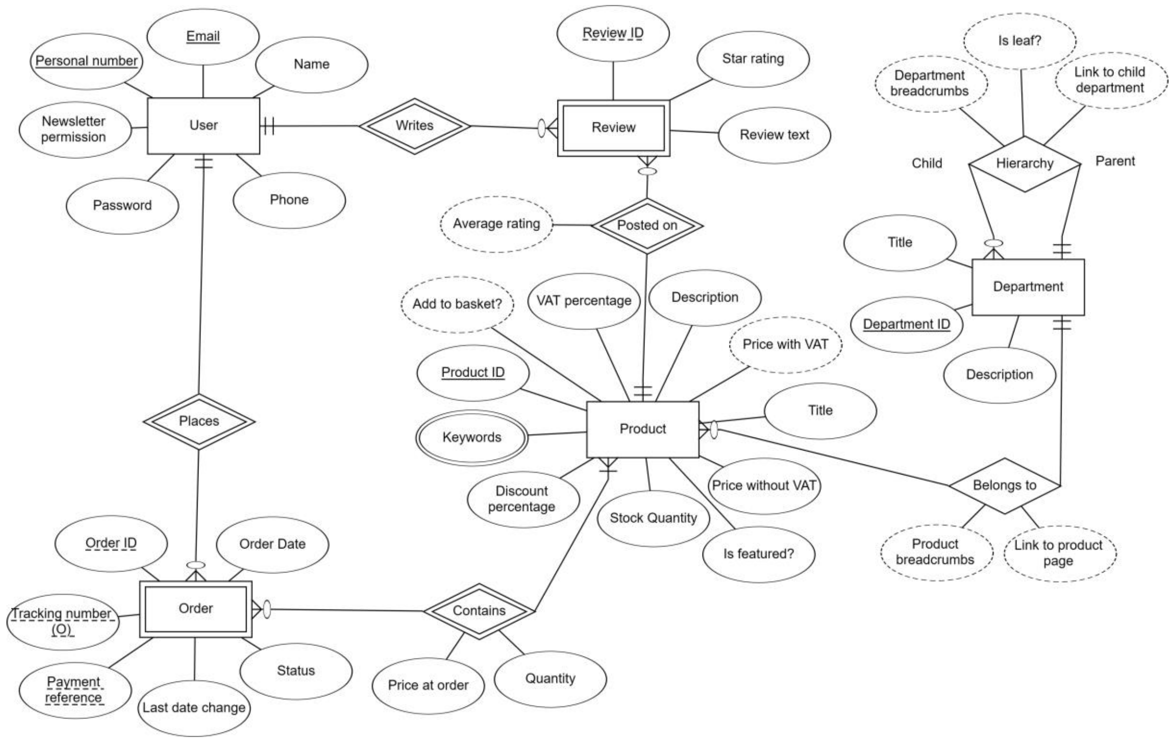


Figure 1.2: Enhanced Entity-Relationship Diagram

1.4 Entities, Attributes, and Relationships

1.4.1 User (strong entity)

- **Attributes:**
  - Personal Number (Unique)
  - Name
  - Email (Unique)
  - Phone
  - Password
  - Newsletter Permission
- **Relationships:**
  - **Writes (identifying relationship):**
    - Cardinality: 1:N (A user can write multiple reviews, but each review is written by one user).
    - Participation: Total for reviews (every review must have an author), Partial for users (not all users write reviews).
  - **Places (identifying relationship):**
    - Cardinality: 1:N (A user can place multiple orders, but each order is placed by one user).

      - Participation: Total for orders (every order must have a user), Partial for users (not all users place orders).

1.4.2 Review (weak entity)

- **Attributes:**
  - Review ID (Unique)
  - Star Rating
  - Review Text
- **Relationships:**
  - **Posted on (identifying relationship):**
    - Attributes:
      - Average rating (Derived)
    - Cardinality: N:1 (A review can be posted on one product, but a product can have multiple reviews).
    - Participation: Total for reviews (every review must be posted on a product), Partial for products (not all products have reviews).
  - **Writes (identifying relationship):**
    - Cardinality: 1:N (A user can write multiple reviews, but each review is written by one user).
    - Participation: Total for reviews (every review must have an author), Partial for users (not all users write reviews).

1.4.3 Product (strong entity)

- **Attributes:**
  - Product ID (Unique)
  - Title
  - Description
  - Price without VAT
  - VAT Percentage
  - Discount Percentage
  - Stock Quantity
  - Keywords (Multivalued)
  - Is featured
- **Derived Attributes:**
  - Price with VAT (Derived): Price + (Price * VAT / 100).
  - Add to Basket (Derived): Enabled if Stock Quantity > 0.
- **Relationships:**
  - **Belongs to:**
    - Attributes:
      - Product Breadcrumbs (Derived): Combines department breadcrumbs

with the prod- uct title.
      - Link to Product Page (Derived): Derived from Product ID and Department ID.
    - Cardinality: N:1 (A product belongs to one department, but a department can have multiple products).
    - Participation: Total for products (every product must belong to a department), Partial for departments (not all departments have products).
  - **Posted on (identifying relationship):** Links products to reviews.
    - Attributes:
      - Average rating (Derived)
    - Cardinality: N:1 (An review can be posted on one product, but a product can have multiple reviews).
    - Participation: Total for reviews (every review must be posted on a product), Partial for products (not all products have reviews).
  - **Contains (identifying relationship):**
    - Attributes:
      - Quantity (Number of products ordered)
      - Price at Order (Product price at the time of order)
    - Cardinality: M:N (An order can contain multiple products, and a product can have multiple orders).
    - Participation: Total for orders (every order must include at least one product), Partial for products (not all products are part of orders).

1.4.4 Department (strong entity)

- **Attributes:**
  - Department ID (Unique)
  - Title
  - Description
- **Relationships:**
  - **Hierarchy (self-referencing):**
    - Attributes:
      - Department Breadcrumbs (Derived): Represents the hierarchical path to the department.
      - Is Leaf (Derived): Indicates if the department has no child departments.
      - Link to Child Department (Derived): URL for navigating to child departments.
    - Cardinality: 1:N (A parent department can have multiple child departments, but a child department has one parent).
    - Participation: Total for children (every child must have a parent), Partial for parents (not all departments are parents).
  - **Belongs to:** Links departments to products.
    - Attributes:
      - Product Breadcrumbs (Derived): Combines department breadcrumbs with the product title.

            - Link to Product Page (Derived): Derived from Product ID and Department ID.
        - Cardinality: N:1 (A product belongs to one department, but a department can have multiple products).
        - Participation: Total for products (every product must belong to a department), Partial for departments (not all departments have products).

1.4.5 Order (weak entity)

- **Attributes:**
    - Order ID (Partially unique)
    - Order Date
    - Status
    - Last Date Change
    - Tracking Number (Partially unique and optional)
    - Payment Reference (Partially Unique)

- **Relationships:**
    - **Contains (identifying relationship):**
        - Attributes:
            - Quantity (Number of products ordered)
            - Price at Order (Product price at the time of order)
        - Cardinality: 1:N (An order can contain multiple products, but a product in the order belongs to one order).
        - Participation: Total for orders (every order must include at least one product), Partial for products (not all products are part of orders).
    - **Places (identifying relationship):**
        - Cardinality: 1:N (A user can place multiple orders, but each order is placed by one user).
        - Participation: Total for orders (every order must have a user), Partial for users (not all users place orders).

1.5 Assumptions

- The **Link to Product Page** is derived from both the Product ID and Department ID.
    - Example format: /departments/{DepartmentID}/products/{ProductID}.
    - For example, a product with ProductID = 101 in DepartmentID = 5 will have the link
      /departments/5/products/101.
- The **Link to Child Department** is derived dynamically based on Department ID.
    - Example format: /departments/{DepartmentID}.
    - For example, a department with DepartmentID = 5 will have the link

/departments/5.

- The **Department Breadcrumbs** dynamically represent the hierarchical structure of departments.
  - Example format: Home > Parent Department > Current Department.
  - For example, a department ”Laptops” under ”Computers and Tablets” will have breadcrumbs
    Home > Computers and Tablets > Laptops.
- The **Product Breadcrumbs** combine the department breadcrumbs with the product title.
  - Example format: Home > Parent Department > Current Department > Product Title.
  - For example, a product ”Mac Book Pro” under ”Laptops” will have breadcrumbs Home > Computers and Tablets > Laptops > MacBook Pro.
- The **Is Leaf** attribute is derived from the self-referencing relationship and identifies whether a department has no children.
- **Keywords** is modeled as a multivalued attribute for products.
- **Logo of the store** is not included in the database as it is a single image file that can be inserted during website design.
- **User ID** attribute is not included in the User entity as it is specified that Email is used for login.
- **Is featured** attribute related to the Products entity is used for generating the featured products table in homepage. Other fields like title, description and current retail price can be fetched from the main Products table dynamically.

Chapter 2: Relational Model & Normalization

2.1 Introduction

This chapter documents the conversion of an Enhanced Entity-Relationship Diagram (EER) into a relational schema and its subsequent normalization into First Normal Form (1NF), Second Normal Form (2NF), and Third Normal Form (3NF). The objective is to eliminate redundancy, resolve dependency anomalies, and ensure consistency while maintaining relational integrity.

2.2 Relational Model

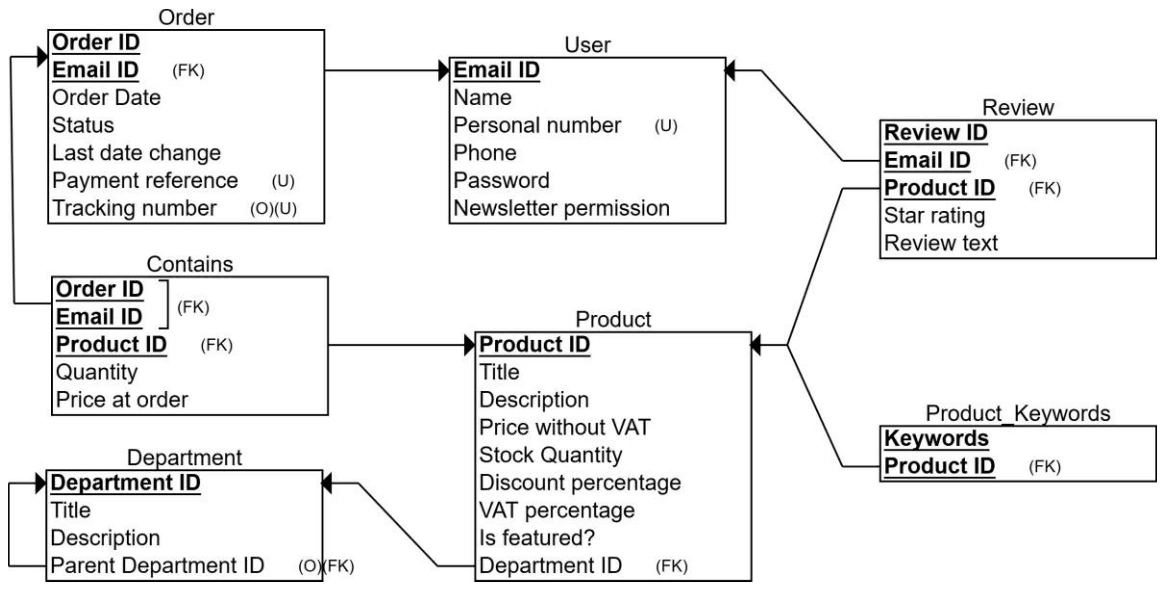


Figure 2.1: Relational Model

2.3 ERD to Relational Model Conversion

The relational schema was derived from the Enhanced Entity-Relationship Diagram (ERD) using the following steps:

- Each strong entity in the ERD was converted into a table with its attributes as columns and its primary key in the relational model.
- Weak entities were converted into tables with composite primary keys, including the primary key of their owner entity as a foreign key.
- Many-to-many relationships were represented using bridge tables (e.g., Contains table for orders and products).
- Attributes of relationships (e.g., Quantity and Price_at Order in the Contains

table) were incorporated into the corresponding bridge tables.

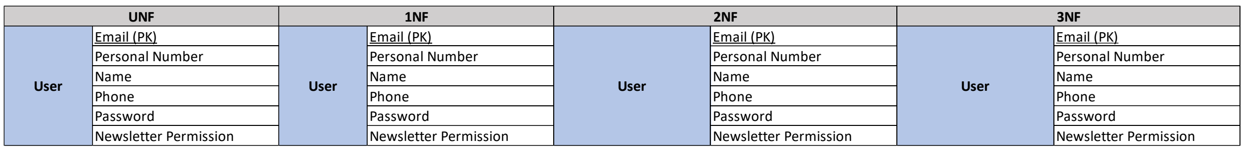
- Derived attributes (e.g., Price with VAT and Product Breadcrumbs) were excluded from the relational model as they can be computed dynamically.
- Multivalued attributes (e.g., Keywords in the Product table) were split into separate tables.

2.4 Normalization Process

2.4.1 User Table

Keys and Functional Dependencies:

- Primary Key: Email_ID
- Functional Dependencies:
  - Email_ID → Personal Number, Name, Phone, Password, Newsletter Permission



UNF		1NF		2NF		3NF	
User	Email (PK)	User	Email (PK)	User	Email (PK)	User	Email (PK)
	Personal Number		Personal Number		Personal Number		Personal Number
	Name		Name		Name		Name
	Phone		Phone		Phone		Phone
	Password		Password		Password		Password
	Newsletter Permission		Newsletter Permission		Newsletter Permission		Newsletter Permission

Figure 2.2: Normalization of User Table

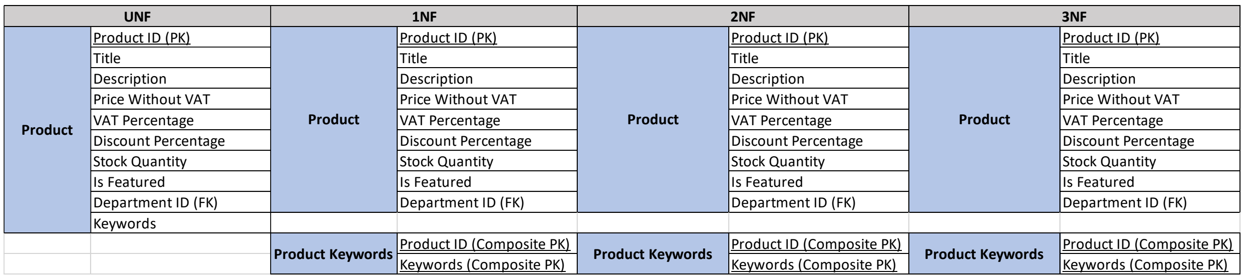
Normalization: The User table was normalized as follows:

- **1NF:** The table already satisfies 1NF since all attributes are single valued.
- **2NF:** The table satisfies 2NF as there are no partial dependencies. All non-prime attributes depend fully on the primary key (Email ID).
- **3NF:** The table satisfies 3NF as there are no transitive dependencies.

2.4.2 Product Table

Keys and Functional Dependencies:

- Primary Key: Product_ID
- Foreign Key: Department ID
- Functional Dependencies:
  - Product_ID → Title, Description, Price without VAT, VAT Percentage, Discount Percentage, Stock Quantity, Is Featured, Department ID, Keywords



UNF		1NF		2NF		3NF	
Product	Product ID (PK)	Product	Product ID (PK)	Product	Product ID (PK)	Product	Product ID (PK)
	Title		Title		Title		Title
	Description		Description		Description		Description
	Price Without VAT		Price Without VAT		Price Without VAT		Price Without VAT
	VAT Percentage		VAT Percentage		VAT Percentage		VAT Percentage
	Discount Percentage		Discount Percentage		Discount Percentage		Discount Percentage
	Stock Quantity		Stock Quantity		Stock Quantity		Stock Quantity
	Is Featured		Is Featured		Is Featured		Is Featured
	Department ID (FK)		Department ID (FK)		Department ID (FK)		Department ID (FK)
	Keywords						
		Product Keywords	Product ID (Composite PK)	Product Keywords	Product ID (Composite PK)	Product Keywords	Product ID (Composite PK)
			Keywords (Composite PK)		Keywords (Composite PK)		Keywords (Composite PK)

Figure 2.3: Normalization of Product Table

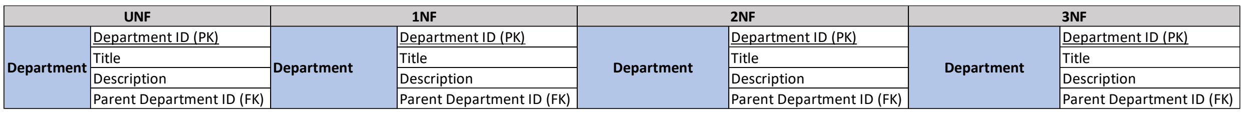
Normalization: The Product table was normalized as follows:

- **1NF:** The Keywords attribute was multivalued. To satisfy 1NF, it was split into a separate Product_Keywords table with a composite primary key (Product_ID, Keyword).
- **2NF:** The table satisfies 2NF as all attributes (Title, Description, etc.) depend fully on the primary key (Product ID).
- **3NF:** The table satisfies 3NF because no transitive dependencies exist.

2.4.3 Department Table

Keys and Functional Dependencies:

- Primary Key: Department_ID
- Foreign Key: Parent Department ID (Self-referencing)
- Functional Dependencies:
  Department ID → Title, Description, Parent Department ID



UNF		1NF		2NF		3NF	
Department	Department ID (PK)	Department	Department ID (PK)	Department	Department ID (PK)	Department	Department ID (PK)
	Title		Title		Title		Title
	Description		Description		Description		Description
	Parent Department ID (FK)		Parent Department ID (FK)		Parent Department ID (FK)		Parent Department ID (FK)

Figure 2.4: Normalization of Department Table

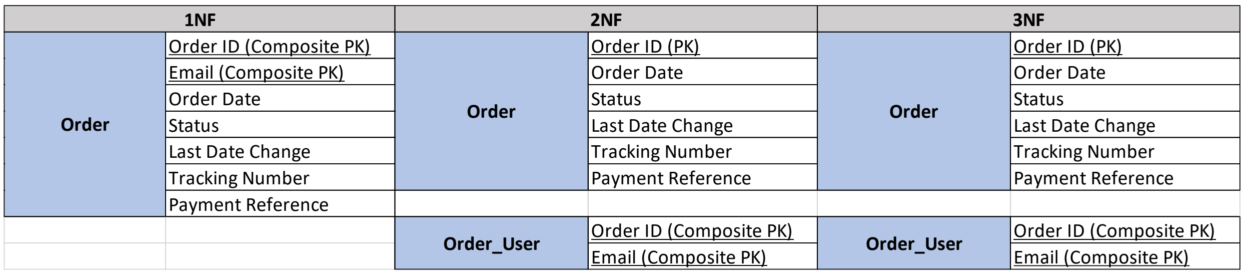
Normalization: The Department table was normalized as follows:

- **1NF:** The table already satisfies 1NF since all attributes are single valued.
- **2NF:** The table satisfies 2NF as all attributes (Title, Description, etc.) depend fully on the primary key (Department_ID).
- **3NF:** The table satisfies 3NF as there are no transitive dependencies. Derived attributes (e.g., Breadcrumbs, Is_Leaf) are not stored but computed dynamically.

2.4.4 Order Table

Keys and Functional Dependencies:

- Composite Primary Key: Order_ID, Email_ID
- Functional Dependencies:
  - Order_ID → Order Date, Status, Tracking_Number, Last Date Change, Payment Reference



1NF		2NF		3NF	
Order	Order ID (Composite PK)	**Order**	Order ID (PK)	**Order**	Order ID (PK)
	Email (Composite PK)		Order Date		Order Date
	Order Date		Status		Status
	Status		Last Date Change		Last Date Change
	Last Date Change		Tracking Number		Tracking Number
	Tracking Number		Payment Reference		Payment Reference
	Payment Reference				
		Order_User	Order ID (Composite PK)	**Order_User**	Order ID (Composite PK)
			Email (Composite PK)		Email (Composite PK)

Figure 2.5: Normalization of Order Table

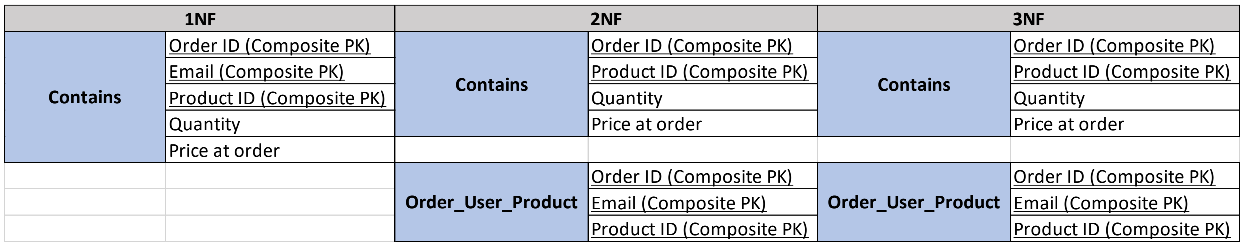
Normalization: The Order table was normalized as follows:

- **1NF:** The table already satisfies 1NF as all attributes are single valued.
- **2NF:** The table violated 2NF because attributes like Order_Date depended only on Order_ID and not on the composite primary key (Order_ID + Email_ID). This was resolved by splitting the table into:
  - Order: Stores attributes dependent on Order ID.
  - Order_User: Tracks the relationship between Order_ID and Email_ID.
- **3NF:** The resulting tables satisfy 3NF because all attributes depend only on their respective primary keys, with no transitive dependencies.

2.4.5 Contains Table

Keys and Functional Dependencies:

- Composite Primary Key: Order_ID, Email_ID, Product_ID
- Functional Dependencies:
  - Order_ID, Product_ID → Quantity, Price_at_Order



1NF		2NF		3NF	
Contains	Order ID (Composite PK)	Contains	Order ID (Composite PK)	Contains	Order ID (Composite PK)
	Email (Composite PK)		Product ID (Composite PK)		Product ID (Composite PK)
	Product ID (Composite PK)		Quantity		Quantity
	Quantity		Price at order		Price at order
	Price at order				
		Order_User_Product	Order ID (Composite PK)	Order_User_Product	Order ID (Composite PK)
			Email (Composite PK)		Email (Composite PK)
			Product ID (Composite PK)		Product ID (Composite PK)

Figure 2.6: Normalization of Contains Table

Normalization: The Contains table was normalized as follows:

- **1NF:** The table already satisfies 1NF as all attributes are single valued.

- **2NF:** The table violated 2NF because Email_ID depended only on Order_ID and not on the composite key (Order ID + Product ID + Email ID). This was resolved by splitting the table into:
  - Order_User_Product: Tracks relationships between Order_ID, Email_ID, and Product_ID.
  - Contains: Stores product-specific details for each order.

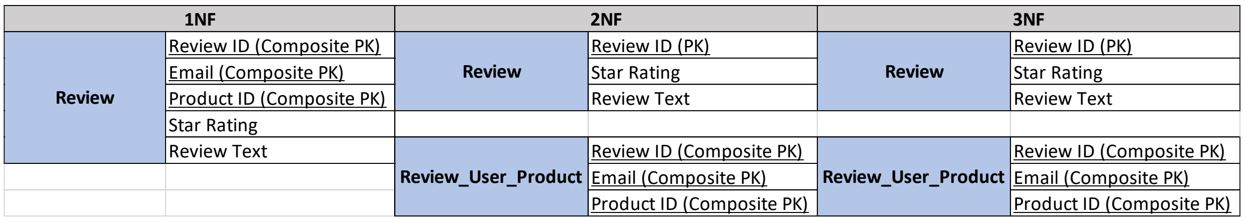
- **3NF:** The resulting tables satisfy 3NF as all attributes depend only on their respective primary keys and no transitive dependencies exist.

2.4.6 Review Table

Keys and Functional Dependencies:

- Composite Primary Key: Email ID, Product_ID, Review_ID

- Functional Dependencies:
  - Review_ID → Star_Rating, Review_Text



1NF		2NF		3NF	
Review	Review ID (Composite PK)	Review	Review ID (PK)	Review	Review ID (PK)
	Email (Composite PK)		Star Rating		Star Rating
	Product ID (Composite PK)		Review Text		Review Text
	Star Rating				
	Review Text	Review_User_Product	Review ID (Composite PK)	Review_User_Product	Review ID (Composite PK)
			Email (Composite PK)		Email (Composite PK)
			Product ID (Composite PK)		Product ID (Composite PK)

Figure 2.7: Normalization of Review Table

Normalization: The Review table was normalized as follows:

- **1NF:** The table already satisfies 1NF as all attributes are single valued.
- **2NF:** The table violated 2NF as attributes like Star_Rating depended only on Review_ID and not the full composite key (Email ID + Product_ID + Review_ID). This was resolved by splitting the table into:
  - Review: Stores attributes dependent on Review ID.
  - Review_User_Product: Tracks relationships between Email ID, Product ID, and Review ID.
- **3NF:** The resulting tables satisfy 3NF as all attributes depend only on their respective primary keys and no transitive dependencies exist.

2.5 Assumptions

- Email ID is added as composite key for Order table as Order is a weak entity and gets the foreign key from owner entity as composite key.
- Contains relationship is an N:M (many to many) relation between Product and Order entities. Hence it was made a separate table with foreign keys from both owner entities as composite keys.
- Email ID is added as composite key for Contains table as Order ID is a partial key and cannot exist individually.
- The Keywords attribute in the Product table is multivalued and stored in a separate table.
- Review is a weak entity. Hence, foreign keys: Email ID and Product ID are added as composite keys from owner entities.
- Derived attributes (e.g., Price with VAT, Product Breadcrumbs, Average rating, etc.) are not stored in the database but computed dynamically.

Chapter 3: SQL Queries and Indices

3.1 Introduction

This chapter documents the SQL code used to generate and populate the finalized tables after the normalization process along with a MySQL reverse engineer diagram attached. This chapter also shows some SQL queries used to fetch data from the database based on the given requirements. Indexing in SQL and how it optimizes the queries in SQL is also discussed along with a brief discussion on the difference between BTree and Hash indices.

3.2 SQL CREATE and INSERT queries

All tables are created in the database using the SQL CREATE command giving necessary keys and constraints and data was inserted using SQL INSERT command as follows:

3.2.1 User Table

```
CREATE TABLE User (
    Email VARCHAR(255) PRIMARY KEY,
    Personal_Number CHAR(12) NOT NULL,
    Name VARCHAR(254) NOT NULL,
    Phone  VARCHAR(12),
    Password VARCHAR(12) NOT NULL,
    CONSTRAINT chk_password_length CHECK (CHAR_LENGTH(Password) BETWEEN
       8 AND 12),
    Newsletter_Permission BOOLEAN DEFAULT FALSE NOT NULL
);

  INSERT INTO User
  (Email, Personal_Number, Name, Phone, Password,
       Newsletter_Permission)
  VALUES
      ('anudeep@gmail.com', '123456789012', 'Anudeep', '1234567890'
           , 'Password1', TRUE),
      ('moutushi@gmail.com', '987654321012', 'Moutushi', '
           0987654321', 'Password2', FALSE),
      ('kokila@gmail.com', '123456789010', 'Kokila', '1234567890','
           Password3', TRUE),
      ('shiva@gmail.com', '987654321010', 'Shiva', '0987654321', '
           Password4', FALSE);
```

3.2.2 Department Table

```
CREATE TABLE Department (
    Department_ID VARCHAR(15) PRIMARY KEY,
    Title VARCHAR(50) NOT NULL UNIQUE,
    Description TEXT NOT NULL,
    Parent_Dept_ID VARCHAR(15),
    FOREIGN KEY (Parent_Dept_ID) REFERENCES Department(Department_ID)
);
```

```
  INSERT INTO Department
  (Department_ID, Title, Description,Parent_Dept_ID)
  VALUES
      ('SD', 'Special Department', 'Grab some good deals
           exclusively at our shop AltOnline !! Happy Shopping!!!}
           ', NULL),
      ('PD001', 'Electronics', 'Top-level department for
           electronics.', NULL),
      ('PD002', 'Home Appliances', 'Top-level department for Home
           Appliances.', NULL),
      ('CD001', 'Computers and Tablets', 'Computers, tablets, and
           related products.', 'PD001'),
      ('CD002', 'TV and Video', 'TVs, projectors, and video
           equipment.', 'PD001'),
      ('CD003', 'Desktops', 'Computers, tablets, and related
           products.', 'CD001'),
      ('CD004', 'Laptops','Lorem ipsum dolor sit amet, consectetur
           adipiscing elit.', 'CD001'),
     ('CD005', 'Tablets','Lorem ipsum dolor sit amet, consectetur
           adipiscing elit.', 'CD001'),
   ('CD006', 'Accessories','Lorem ipsum dolor sit amet, consectetur
   adipiscing elit.', 'CD001');
```

3.2.3 Orders Table

```
CREATE TABLE Orders (
        Order_ID VARCHAR(15) PRIMARY KEY,
        Order_Date DATE NOT NULL,
        Order_Status ENUM('New', 'Open', 'Dispatched') NOT NULL,
        Last_Change_Date DATE,
        Tracking_Number VARCHAR(20) UNIQUE, Payment_Reference
        VARCHAR(20) UNIQUE NOT NULL
);

 INSERT INTO Orders
 (Order_ID, Order_Date, Order_Status,Last_Change_Date, Tracking_Number,
     Payment_Reference)
 VALUES
('0001', '2024-12-01', 'New', '2024-12-01', 'TRACK12345', 'PAY12345');
```

3.2.4 Order_User Table

```
CREATE TABLE Order_User (
        Order_ID VARCHAR(15),
        Email VARCHAR(255),
        PRIMARY KEY(Order_ID, Email),
        FOREIGN KEY(Order_ID) REFERENCES Orders(Order_ID),
        FOREIGN KEY(Email) REFERENCES User(Email)
);

 INSERT INTO Order_User (Order_ID, Email)
 VALUES
        ('0001', 'anudeep@gmail.com');
```

3.2.5 Product Table

```sql
CREATE TABLE Product (
    ProductID VARCHAR(15) PRIMARY KEY,
    Title VARCHAR(50) NOT NULL UNIQUE,
    Description TEXT NOT NULL,
    PriceWithoutVAT DECIMAL(10, 2) NOT NULL CHECK (PriceWithoutVAT >= 0),
    VATPercentage DECIMAL(5, 2) NOT NULL CHECK (VATPercentage > 0.00),
    DiscountPercentage DECIMAL(5, 2) NOT NULL,
    StockQuantity INT UNSIGNED NOT NULL,
    IsFeatured BOOLEAN DEFAULT FALSE,
    Department_ID VARCHAR(15),
    FOREIGN KEY (Department_ID) REFERENCES Department(Department_ID)
);
```

```sql
  INSERT INTO Product
  (ProductID, Title, Description, PriceWithoutVAT, VATPercentage,
        DiscountPercentage, StockQuantity, IsFeatured,
        Department_ID)
  VALUES
      ('P001', 'Wireless Mouse', 'Lorem ipsum dolor sit amet, consectetur
            adipiscing elit.', 29.99, 20.00, 5.00, 100, 0,'CD006'),
      ('P002', 'Gaming Keyboard', 'Lorem ipsum dolor sit amet,
            consectetur adipiscing elit.', 49.99, 20.00, 10.00, 50, 1, '
            CD006'),
      ('P003', 'Bluetooth Speaker', 'Lorem ipsum dolor sit amet,
            consectetur adipiscing elit.', 39.99, 20.00, 7.00, 200, 0, '
            CD006'),
      ('P004', 'Macbook', 'Lorem ipsum dolor sit amet, consectetur
            adipiscing elit.', 99.99, 20.00, 15.00, 75, 1, 'CD004'),
      ('P005', 'Surface Pro', 'Lorem ipsum dolor sit amet,
            consectetur adipiscing elit.', 129.99, 20.00, 5.00, 120,
            0, 'CD004'),
      ('P006', 'Samsung Tab', 'Lorem ipsum dolor sit amet,
            consectetur adipiscing elit.', 19.99, 20.00, 0.00, 150, 0,
            'CD005'),
      ('P007', 'iPad', 'Lorem ipsum dolor sit amet, consectetur adipiscing
            elit.', 79.99, 20.00, 10.00, 80, 1, 'CD005'),
      ('P008', 'Asus laptop', 'Lorem ipsum dolor sit amet,
            consectetur adipiscing elit.', 299.99, 20.00, 20.00, 40
            , 1, 'CD004'),
      ('P009', 'USB-C Hub', 'Lorem ipsum dolor sit amet,
            consectetur adipiscing elit.', 34.99, 20.00, 8.00, 300, 0,
            'CD006'),
      ('P010', 'Smartphone Case', 'Lorem ipsum dolor sit amet,
            consectetur adipiscing elit.', 14.99, 20.00, 0.00, 500, 0, '
            CD006');
```

3.2.6 Product_Keywords Table

```sql
  CREATE TABLE Product_Keywords (
        ProductID VARCHAR(15) NOT NULL,
        Keyword VARCHAR(15) NOT NULL,
        PRIMARY KEY (ProductID, Keyword),
        FOREIGN KEY (ProductID) REFERENCES Product(ProductID)
  );
```

```
INSERT INTO Product_Keywords (ProductID, Keyword)
VALUES
     ('P001', 'Wireless'),
     ('P001', 'Mouse'),
     ('P002', 'Keyboard'),
     ('P002', 'Gaming'),
     ('P003', 'Bluetooth'),
     ('P002', 'Mouse'),
     ('P003', 'Mouse');
```

3.2.7 Review Table

```
CREATE TABLE Review (
     Review_ID VARCHAR(15) PRIMARY KEY,
     Star_Rating TINYINT UNSIGNED NOT NULL,
     CHECK (Star_Rating BETWEEN 1 AND 5),
     Review_Text  TEXT
);


INSERT INTO Review (Review_ID, Star_Rating, Review_Text)
VALUES
     ('R001', 5, 'Excellent product, highly recommended!'),
     ('R002', 4, 'Good value for money, but delivery was delayed.'),
     ('R003', 3, 'Average quality, expected better for the price.'),
     ('R004', 2, 'Not satisfied, the product broke within a week.'),
     ('R005', 1, 'Terrible experience, the product was defective.');
```

3.2.8 Review_User_Product Table

```
CREATE TABLE Review_User_Product (
        Review_ID VARCHAR(15) NOT NULL,
        Email VARCHAR(255) NOT NULL,
        ProductID VARCHAR(15) NOT NULL,
        PRIMARY KEY (Review_ID, Email, ProductID),
        FOREIGN KEY (Review_ID) REFERENCES Review(Review_ID),
        FOREIGN KEY (Email) REFERENCES User(Email),
        FOREIGN KEY (ProductID) REFERENCES Product(ProductID)
);

INSERT INTO Review_User_Product (Review_ID, Email, ProductID)
VALUES
     ('R001', 'anudeep@gmail.com', 'P001'),
     ('R002', 'anudeep@gmail.com', 'P002'),
     ('R003', 'shiva@gmail.com', 'P001'),
     ('R004', 'moutushi@gmail.com', 'P004'),
     ('R005', 'kokila@gmail.com', 'P005');
```

3.2.9 Contains Table

```
 CREATE TABLE Contains (
      Order_ID VARCHAR(15),
      ProductID VARCHAR(15),
      Quantity INT UNSIGNED NOT NULL,
      CHECK (Quantity > 0),
      Price_at_order DECIMAL(10,2) NOT NULL,
      CHECK (Price_at_order >=0),
      PRIMARY KEY (Order_ID, ProductID),
      FOREIGN KEY (Order_ID) REFERENCES Orders(Order_ID),
      FOREIGN KEY (ProductID) REFERENCES Product(ProductID)
);

INSERT INTO Contains (Order_ID, ProductID, Quantity, Price_at_order)
VALUES
      ('O001', 'P002', 3, 40.00);
```

3.2.10 Order_User_Product Table

```
CREATE TABLE Order_User_Product (
     Order_ID VARCHAR(15) NOT NULL,
     Email VARCHAR(255) NOT NULL,
     ProductID VARCHAR(15) NOT NULL,
     PRIMARY KEY (Order_ID, Email, ProductID),
     FOREIGN KEY (Order_ID) REFERENCES Orders(Order_ID),
     FOREIGN KEY (Email) REFERENCES User(Email),
     FOREIGN KEY (ProductID) REFERENCES Product(ProductID)
);

INSERT INTO Order_User_Product (Order_ID, Email, ProductID)
VALUES
      ('O001', 'anudeep@gmail.com', 'P002');
```

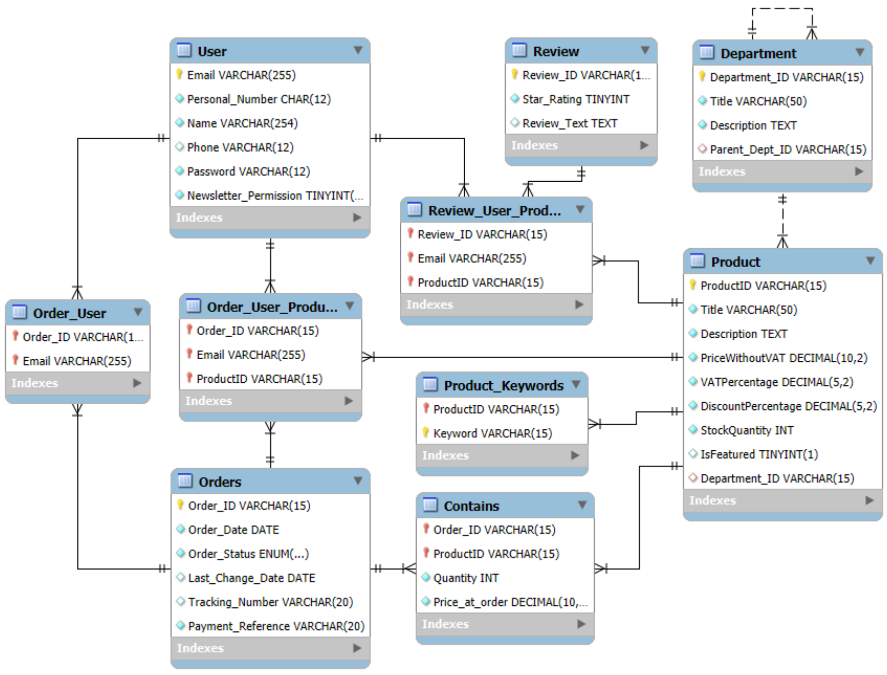
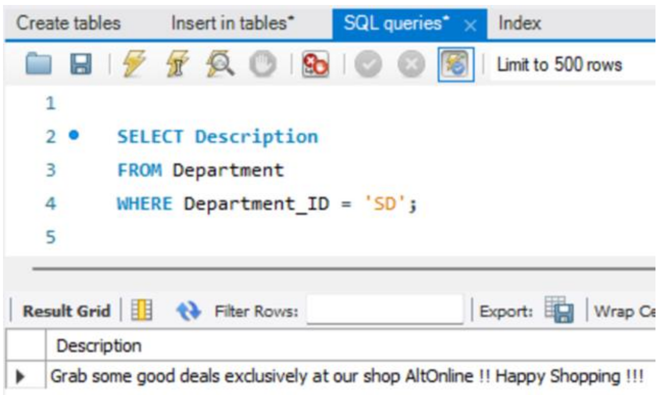


Figure 3.1: MySQL Workbench Reverse Engineer Diagram

3.3 SQL queries to fetch required data

Query 1: Welcome text for the homepage

This query retrieves welcome text for homepage stored in the special department in department table.

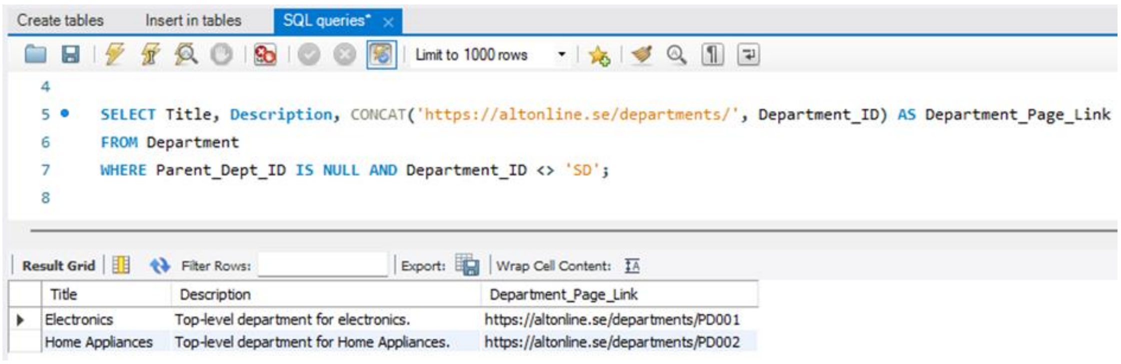


```sql
SELECT Description
FROM Department
WHERE Department_ID = 'SD';
```

Description
Grab some good deals exclusively at our shop AltOnline !! Happy Shopping !!!

Query 2: List of the top level departments with fields needed for the homepage

This query retrieves the list of top level departments (excluding the special department) along with the fields needed for homepage (Title, Description, Department Page Link) calculating the values dynamically.

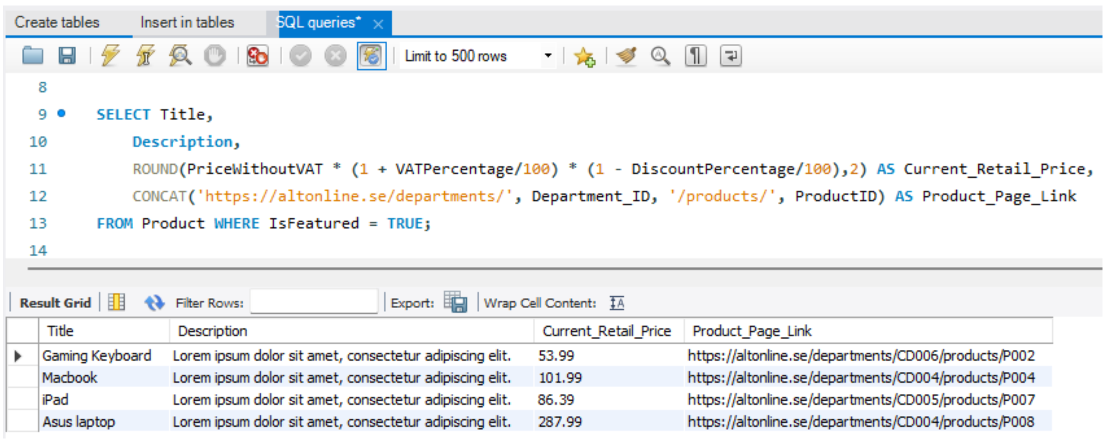


```sql
SELECT Title, Description, CONCAT('https://altonline.se/departments/', Department_ID) AS Department_Page_Link
FROM Department
WHERE Parent_Dept_ID IS NULL AND Department_ID <> 'SD';
```

Title	Description	Department_Page_Link
Electronics	Top-level department for electronics.	https://altonline.se/departments/PD001
Home Appliances	Top-level department for Home Appliances.	https://altonline.se/departments/PD002

Query 3: List of the featured products with fields needed for the homepage

This query retrieves the list of the featured products along with the fields needed for homepage (Title, Description, Current Retail Price, Product Page Link) calculating the values dynamically.

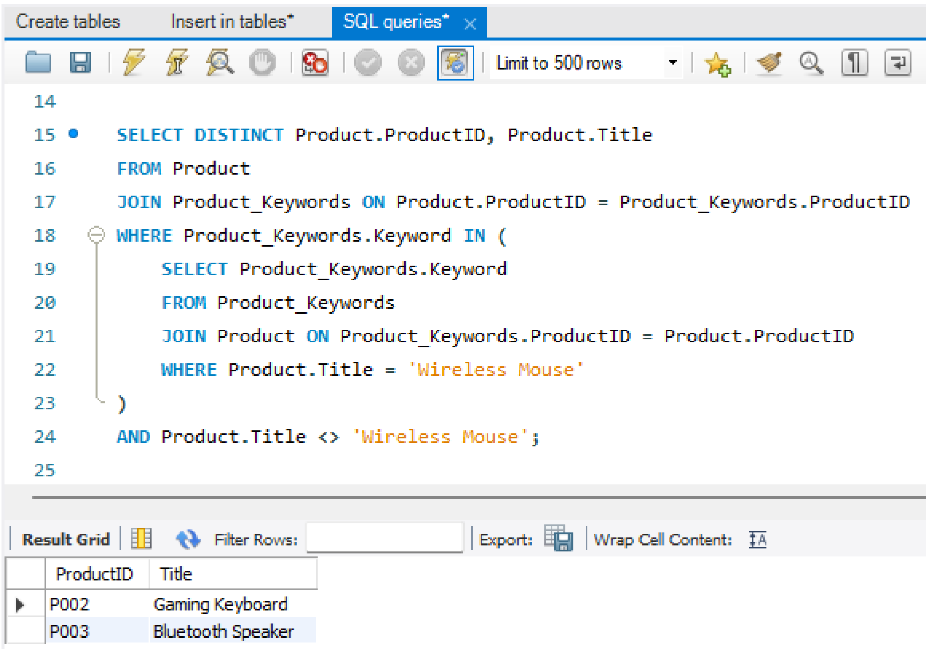


```sql
SELECT Title,
    Description,
    ROUND(PriceWithoutVAT * (1 + VATPercentage/100) * (1 - DiscountPercentage/100),2) AS Current_Retail_Price,
    CONCAT('https://altonline.se/departments/', Department_ID, '/products/', ProductID) AS Product_Page_Link
FROM Product WHERE IsFeatured = TRUE;
```

Title	Description	Current_Retail_Price	Product_Page_Link
Gaming Keyboard	Lorem ipsum dolor sit amet, consectetur adipiscing elit.	53.99	https://altonline.se/departments/CD006/products/P002
Macbook	Lorem ipsum dolor sit amet, consectetur adipiscing elit.	101.99	https://altonline.se/departments/CD004/products/P004
iPad	Lorem ipsum dolor sit amet, consectetur adipiscing elit.	86.39	https://altonline.se/departments/CD005/products/P007
Asus laptop	Lorem ipsum dolor sit amet, consectetur adipiscing elit.	287.99	https://altonline.se/departments/CD004/products/P008

Query 4: Given a product, list all keyword-related products

This query retrieves the list of all keyword related products given a product by joining the product and product keywords tables and using nested query.

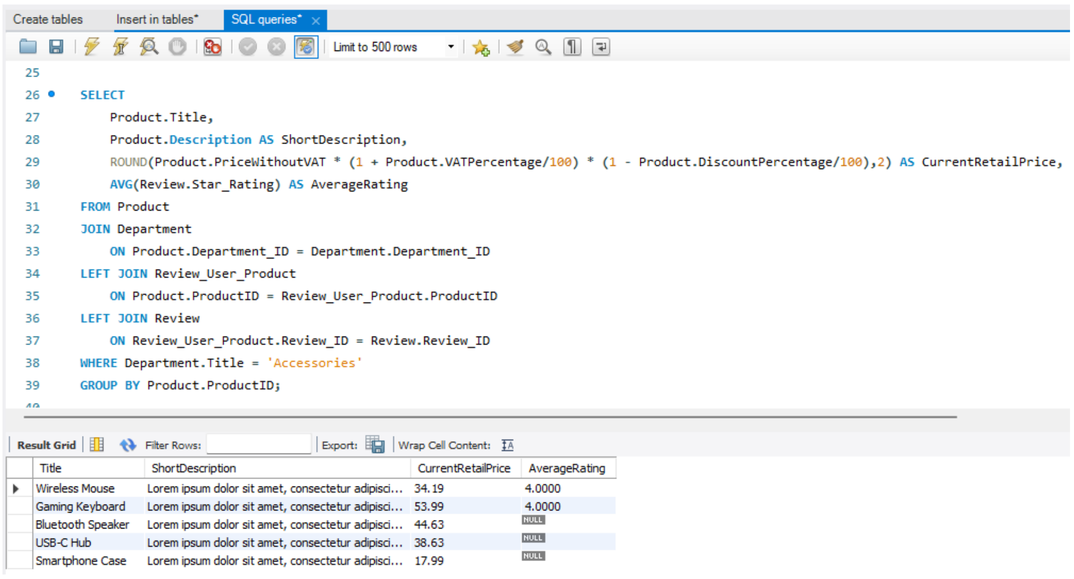


```sql
SELECT DISTINCT Product.ProductID, Product.Title
FROM Product
JOIN Product_Keywords ON Product.ProductID = Product_Keywords.ProductID
WHERE Product_Keywords.Keyword IN (
    SELECT Product_Keywords.Keyword
    FROM Product_Keywords
    JOIN Product ON Product_Keywords.ProductID = Product.ProductID
    WHERE Product.Title = 'Wireless Mouse'
)
AND Product.Title <> 'Wireless Mouse';
```

ProductID	Title
P002	Gaming Keyboard
P003	Bluetooth Speaker

Query 5: Given a department, list of all its products (title, short description, current retail price) with their average rating

This query retrieves the list of all products (title, short description, current retail price) with their average rating given a department by joining the Product, Department, Review and Review_User_Product tables and using left join as products without any ratings also need to be included.

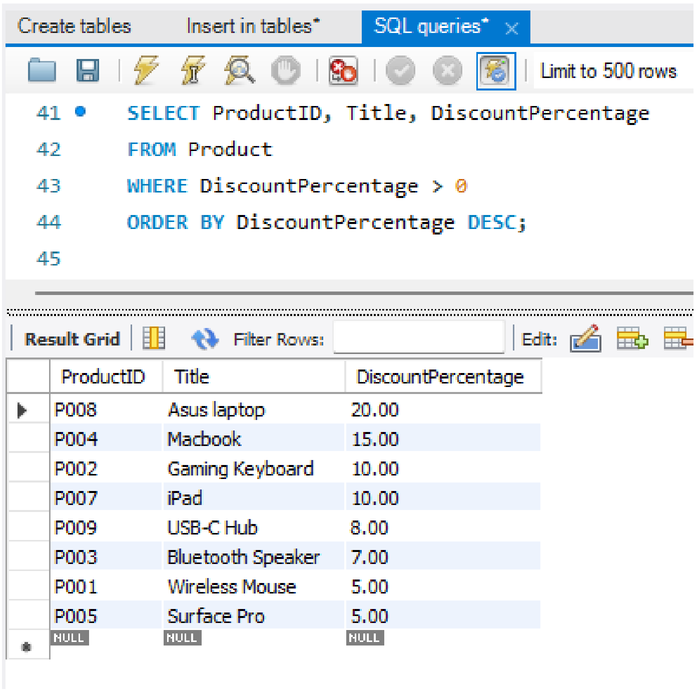


```sql
SELECT
    Product.Title,
    Product.Description AS ShortDescription,
    ROUND(Product.PriceWithoutVAT * (1 + Product.VATPercentage/100) * (1 - Product.DiscountPercentage/100),2) AS CurrentRetailPrice,
    AVG(Review.Star_Rating) AS AverageRating
FROM Product
JOIN Department
    ON Product.Department_ID = Department.Department_ID
LEFT JOIN Review_User_Product
    ON Product.ProductID = Review_User_Product.ProductID
LEFT JOIN Review
    ON Review_User_Product.Review_ID = Review.Review_ID
WHERE Department.Title = 'Accessories'
GROUP BY Product.ProductID;
```

Title	ShortDescription	CurrentRetailPrice	AverageRating
Wireless Mouse	Lorem ipsum dolor sit amet, consectetur adipisci...	34.19	4.0000
Gaming Keyboard	Lorem ipsum dolor sit amet, consectetur adipisci...	53.99	4.0000
Bluetooth Speaker	Lorem ipsum dolor sit amet, consectetur adipisci...	44.63	NULL
USB-C Hub	Lorem ipsum dolor sit amet, consectetur adipisci...	38.63	NULL
Smartphone Case	Lorem ipsum dolor sit amet, consectetur adipisci...	17.99	NULL

Query 6: List of all products on sale sorted by the discount percentage (starting with the biggest discount)

This query retrieves the list of all products on sale sorted by the discount percentage (starting with the biggest discount) using the order by command.



```sql
SELECT ProductID, Title, DiscountPercentage
FROM Product
WHERE DiscountPercentage > 0
ORDER BY DiscountPercentage DESC;
```

ProductID	Title	DiscountPercentage
P008	Asus laptop	20.00
P004	Macbook	15.00
P002	Gaming Keyboard	10.00
P007	iPad	10.00
P009	USB-C Hub	8.00
P003	Bluetooth Speaker	7.00
P001	Wireless Mouse	5.00
P005	Surface Pro	5.00
NULL	NULL	NULL

3.4 SQL queries to create indices

Two of the above queries are optimized by adding indices and the number of rows scanned was compared before and after adding the indices using the EXPLAIN command.

Query 1: List of the featured products with fields needed for the homepage

```
EXPLAIN SELECT
     Title,
     Description,
     ROUND(PriceWithoutVAT * (1 + VATPercentage/100) *
     (1- DiscountPercentage/100),2) AS Current_Retail_Price,
      CONCAT('https://altonline.se/departments/',
      Department_ID, '/products/', ProductID) AS Product_Page_Link
FROM Product
WHERE IsFeatured = TRUE;

CREATE INDEX idx_IsFeatured ON Product (IsFeatured);
SHOW INDEX FROM Product;
```

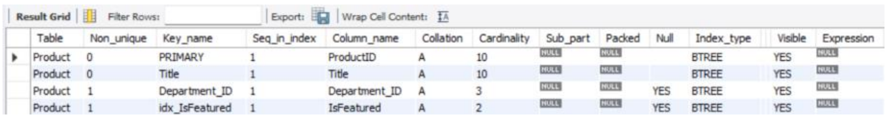
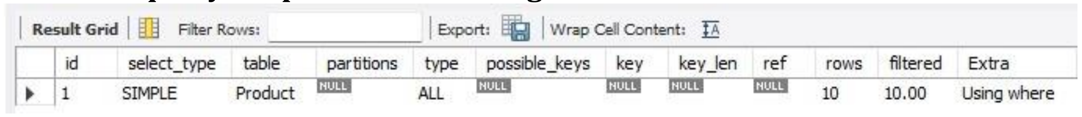


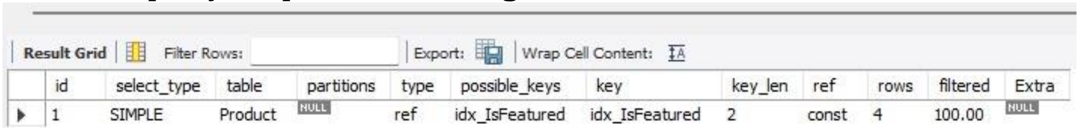
Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Visible	Expression
Product	0	PRIMARY	1	ProductID	A	10	NULL	NULL		BTREE	YES	NULL
Product	0	Title	1	Title	A	10	NULL	NULL		BTREE	YES	NULL
Product	1	Department_ID	1	Department_ID	A	3	NULL	NULL	YES	BTREE	YES	NULL
Product	1	idx_IsFeatured	1	IsFeatured	A	2	NULL	NULL	YES	BTREE	YES	NULL

EXPLAIN query output before adding Index:



id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	Product	NULL	ALL	NULL	NULL	NULL	NULL	10	10.00	Using where

EXPLAIN query output after adding Index:



id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	Product	NULL	ref	idx_IsFeatured	idx_IsFeatured	2	const	4	100.00	NULL

This query is optimized by adding an index on IsFeatured column of the Product table.

- **idx_IsFeatured**: Speeds up filtering of rows where IsFeatured = TRUE by avoiding a full table scan.

The output shows that the number of rows scanned has decreased from 10 to 4 in the rows column and the filtered percentage has increased from 10 to 100 improving the efficiency of the query.

Query 2: Given a product, list all keyword-related products

```
EXPLAIN SELECT DISTINCT Product.ProductID, Product.Title
FROM Product
JOIN Product_Keywords
ON Product.ProductID = Product_Keywords.ProductID
WHERE Product_Keywords.Keyword IN
      (SELECT Product_Keywords.Keyword
      FROM Product_Keywords
      JOIN Product ON Product_Keywords.ProductID = Product.ProductID
      WHERE Product.Title = 'Wireless Mouse')
AND Product.Title <> 'Wireless Mouse';
```

```sql
CREATE INDEX idx_product_keywords_keyword ON
Product_Keywords(Keyword);

CREATE INDEX idx_product_keywords_productid_keyword ON
Product_Keywords(ProductID, Keyword);

SHOW INDEX FROM  Product_Keywords;
```

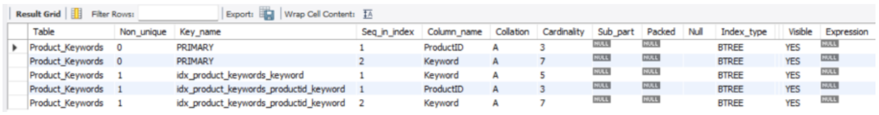


Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Visible	Expression
Product_Keywords	0	PRIMARY	1	ProductID	A	3	NULL	NULL		BTREE	YES	NULL
Product_Keywords	0	PRIMARY	2	Keyword	A	7	NULL	NULL		BTREE	YES	NULL
Product_Keywords	1	idx_product_keywords_keyword	1	Keyword	A	5	NULL	NULL		BTREE	YES	NULL
Product_Keywords	1	idx_product_keywords_productid_keyword	1	ProductID	A	3	NULL	NULL		BTREE	YES	NULL
Product_Keywords	1	idx_product_keywords_productid_keyword	2	Keyword	A	7	NULL	NULL		BTREE	YES	NULL

```sql
CREATE INDEX idx_product_title ON  Product(Title);

SHOW INDEX FROM  Product;
```

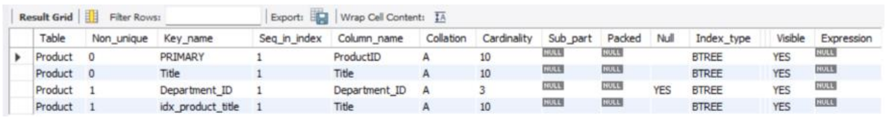
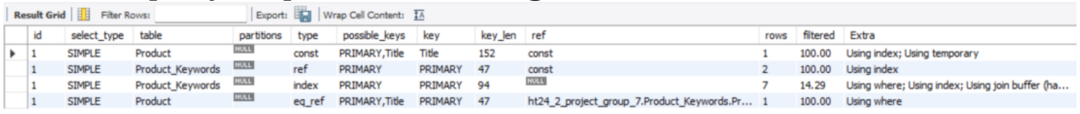


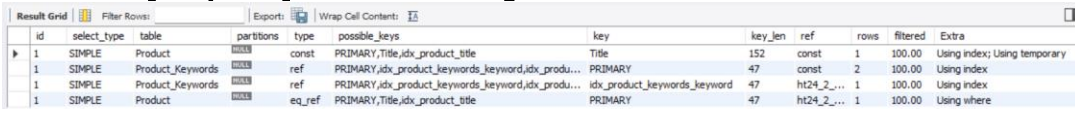
Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Visible	Expression
Product	0	PRIMARY	1	ProductID	A	10	NULL	NULL		BTREE	YES	NULL
Product	0	Title	1	Title	A	10	NULL	NULL		BTREE	YES	NULL
Product	1	Department_ID	1	Department_ID	A	3	NULL	NULL	YES	BTREE	YES	NULL
Product	1	idx_product_title	1	Title	A	10	NULL	NULL		BTREE	YES	NULL

EXPLAIN query output before adding Indices:



id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	Product	NULL	const	PRIMARY,Title	Title	152	const	1	100.00	Using index; Using temporary
1	SIMPLE	Product_Keywords	NULL	ref	PRIMARY	PRIMARY	47	const	2	100.00	Using index
1	SIMPLE	Product_Keywords	NULL	index	PRIMARY	PRIMARY	94	NULL	7	14.29	Using where; Using index; Using join buffer (ha...
1	SIMPLE	Product	NULL	eq_ref	PRIMARY,Title	PRIMARY	47	ht24_2_project_group_7.Product_Keywords.Pr...	1	100.00	Using where

EXPLAIN query output after adding Indices:



id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	Product	NULL	const	PRIMARY,Title,idx_product_title	Title	152	const	1	100.00	Using index; Using temporary
1	SIMPLE	Product_Keywords	NULL	ref	PRIMARY,idx_product_keywords_keyword,idx_produ...	PRIMARY	47	const	2	100.00	Using index
1	SIMPLE	Product_Keywords	NULL	ref	PRIMARY,idx_product_keywords_keyword,idx_produ...	idx_product_keywords_keyword	47	ht24_2_...	1	100.00	Using index
1	SIMPLE	Product	NULL	eq_ref	PRIMARY,Title,idx_product_title	PRIMARY	47	ht24_2_...	1	100.00	Using where

This query is optimized by adding the following indices on the Keyword column of Product_Keywords table, Title column of Product table and a composite index on the ProductID and Keyword columns of Product_Keywords table.

- **idx_product_keywords_keyword**: Improves the lookup of keywords used in the filtering condition.
- **idx_product_keywords_productid_keyword**: Optimizes the join between Product and Product_Keywords, which is critical for improving overall performance.
- **idx_product_title**: Makes searching for specific products by title more efficient in the subquery.

The output (row 3) shows that the number of rows scanned has decreased from 7 to 1 in the rows column and the filtered percentage has increased from 14.29 to 100 improving the efficiency of the query.

3.5 BTree and Hash indices

BTree structure: It's a default structure used by MySQL while creating an index for a required column in a table. It follows a balanced tree like structure for storing index on the required column. Its performance is efficient while using queries with WHERE clause for range (BETWEEN, <,>, etc), equality (=) and sorting (ORDER BY).

Hash structure: It's used mainly in memory tables where a key-value pair structure is followed, while creating index in the required table. Hash structure is supported only for equality queries (= in WHERE clause) and is not supported for using range & sorting queries. This structure name has to be explicitly mentioned while creating index for the required column in a table.

Feature	**BTree Index Structure**	**Hash Index Structure**
Operations supported	It supports for using range queries (BETWEEN, <,> in WHERE clause) sorting queries (ORDER BY) and equality queries (= in WHERE clause).	It supports only for using equality queries (= in WHERE clause). It does not support for using range and sorting queries.
Performance	Efficient for using range queries.	Fast for exact match queries (= in WHERE clause).
Structure followed	Balanced tree like structure	Key-value pair structure
Usage	Commonly used for general purpose indexing.	Commonly using for querying the tables with exact matches (eg: lookup tables)
Memory	Requires more memory due to tree structure and pointers.	Requires less memory as it stores only the hash table.
Data storage order	Data is stored in a sorted order, which helps in range queries.	Data is stored as hashed keys, order is not maintained.
Default index structure type	It's the default index structure type used while creating index for a required column in a table.	Its not a default index structure type. Its not used unless explicitly mentioned while creating index in a table.
MySQL Storage Engine	Supported by most storage engines (e.g., InnoDB, MyISAM).	Supported only by the MEMORY storage engine.

Table 3.1: Differences between BTree and Hash indices

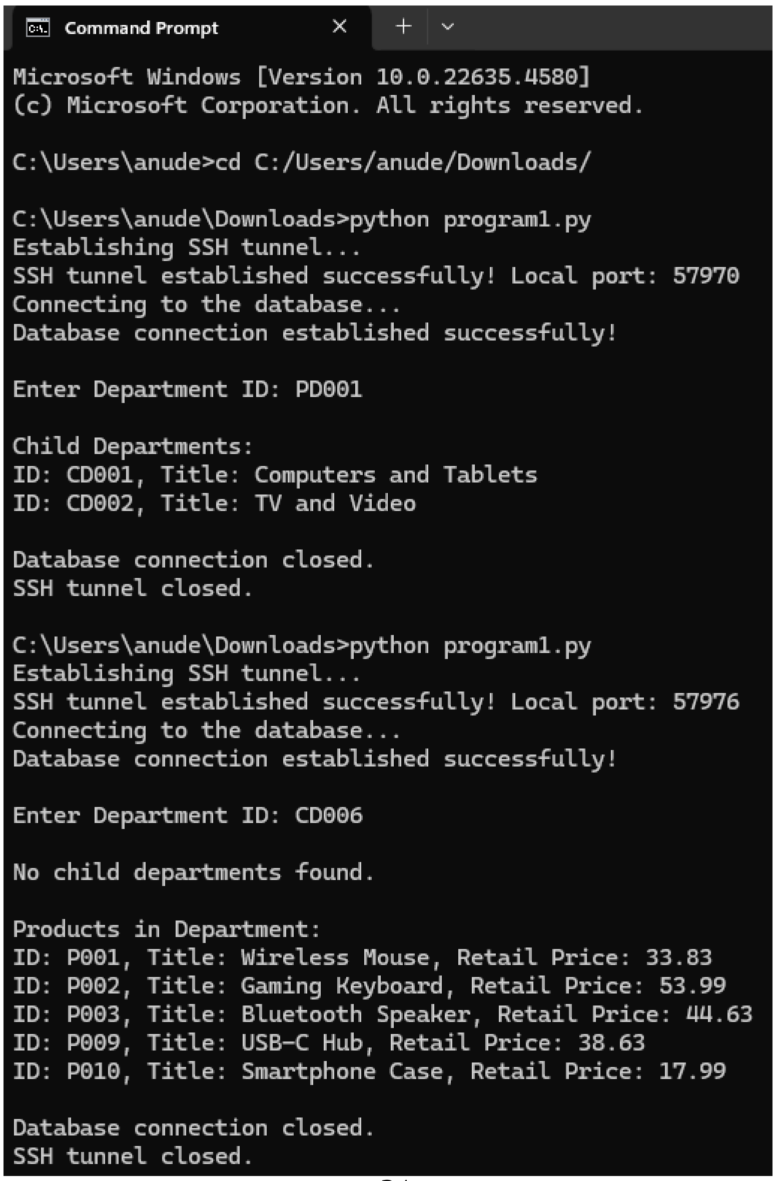
Chapter 4: Connecting to Database using PyMySQL

4.1 Introduction

This chapter documents the creation of Python programs to connect to the created database using PyMySQL and fetch the required data using SQL queries. The programs were run and the outputs were attached along with the code used. The instructions to run the programs in a local machine were also added.

4.2 Program-1: List Products or Departments

A Python program was created which connects to the database, asks the user for a department ID (i.e., the value of the primary key) and lists all its products (outputting the ID, the title and the retail price after the discount) if the given department is a leaf department, otherwise lists all its child departments (outputting the ID and the title). The code can be found in the section 4.4.

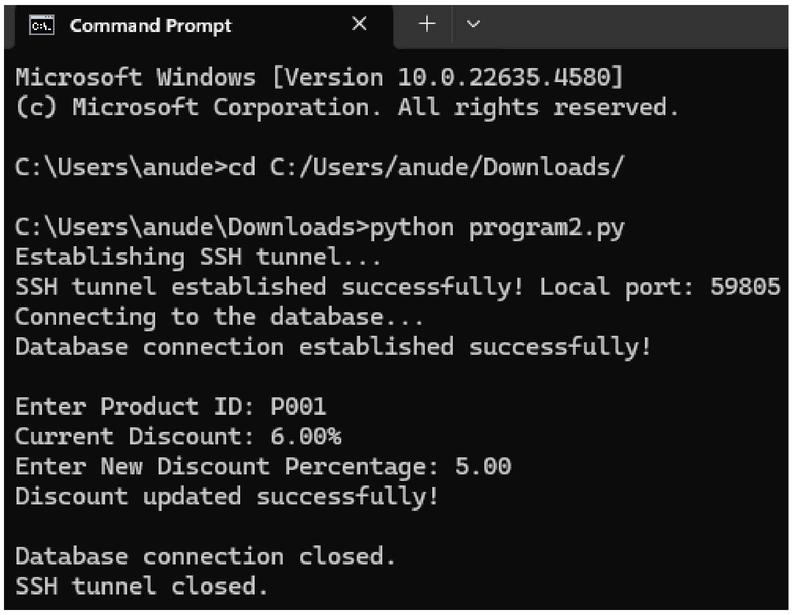


```
Microsoft Windows [Version 10.0.22635.4580]
(c) Microsoft Corporation. All rights reserved.

C:\Users\anude>cd C:/Users/anude/Downloads/

C:\Users\anude\Downloads>python program1.py
Establishing SSH tunnel...
SSH tunnel established successfully! Local port: 57970
Connecting to the database...
Database connection established successfully!

Enter Department ID: PD001

Child Departments:
ID: CD001, Title: Computers and Tablets
ID: CD002, Title: TV and Video

Database connection closed.
SSH tunnel closed.

C:\Users\anude\Downloads>python program1.py
Establishing SSH tunnel...
SSH tunnel established successfully! Local port: 57976
Connecting to the database...
Database connection established successfully!

Enter Department ID: CD006

No child departments found.

Products in Department:
ID: P001, Title: Wireless Mouse, Retail Price: 33.83
ID: P002, Title: Gaming Keyboard, Retail Price: 53.99
ID: P003, Title: Bluetooth Speaker, Retail Price: 44.63
ID: P009, Title: USB-C Hub, Retail Price: 38.63
ID: P010, Title: Smartphone Case, Retail Price: 17.99

Database connection closed.
SSH tunnel closed.
```

In the above screenshot, the program was run in the command prompt of our local machine which is connecting to the database and the Department ID: PD001 was input which belongs to the non-leaf department: Electronics. So, the list of child departments was displayed. In the second section, the program was run again and the Department ID: CD006 was input which belongs to the leaf department: Accessories. So, the list of products in the department were displayed.

4.3 Program-1: Update Product Discount

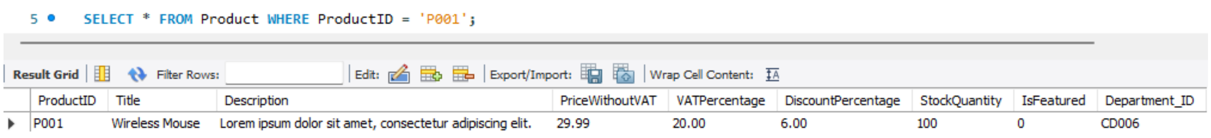
A Python program was created which connects to the database, asks for a product ID, shows the current discount and allows the user to change it.



```
Microsoft Windows [Version 10.0.22635.4580]
(c) Microsoft Corporation. All rights reserved.

C:\Users\anude>cd C:/Users/anude/Downloads/

C:\Users\anude\Downloads>python program2.py
Establishing SSH tunnel...
SSH tunnel established successfully! Local port: 59805
Connecting to the database...
Database connection established successfully!

Enter Product ID: P001
Current Discount: 6.00%
Enter New Discount Percentage: 5.00
Discount updated successfully!

Database connection closed.
SSH tunnel closed.
```

In the above screenshot, the program was run in the command prompt of our local machine which is connecting to the database and the Product ID: P001 was input which belongs to the Product: Wireless Mouse. So, the current discount percentage is displayed as 6%. The new discount percentage was input as 5% and it was updated in the database. SQL queries were run before and after running the program to check the update.

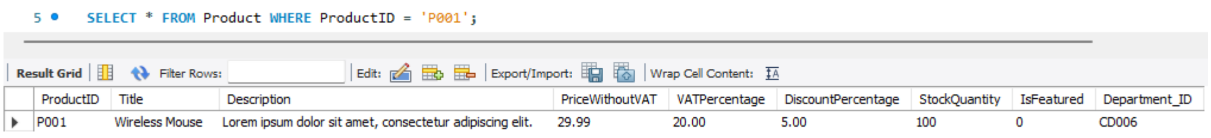
Before running the program:



```
5 •  SELECT * FROM Product WHERE ProductID = 'P001';
```

ProductID	Title	Description	PriceWithoutVAT	VATPercentage	DiscountPercentage	StockQuantity	IsFeatured	Department_ID
P001	Wireless Mouse	Lorem ipsum dolor sit amet, consectetur adipiscing elit.	29.99	20.00	6.00	100	0	CD006

After running the program:



```
5 •  SELECT * FROM Product WHERE ProductID = 'P001';
```

ProductID	Title	Description	PriceWithoutVAT	VATPercentage	DiscountPercentage	StockQuantity	IsFeatured	Department_ID
P001	Wireless Mouse	Lorem ipsum dolor sit amet, consectetur adipiscing elit.	29.99	20.00	5.00	100	0	CD006

4.4 Instructions for Running the Code

Follow these steps to run the provided Python programs:

- **Prerequisites:**
  - Install Python version 3.7 or later.
  - Install the required libraries using the following command:

```
        pip install paramiko sshtunnel pymysql
```

  - Ensure you have valid credentials for the SSH server and MySQL database.
- **Verify Credentials:**
  - SSH Server: `fries.it.uu.se`, Port: `22`
  - SSH Username: , Password: `Your Password`
  - MySQL Server: `ht24_2_project_group_7`, Username: `ht24_2_group_7`, Password: `pasSWd_7`
- **Execution:**
  - Save the program to a file, e.g., `program.py`.
  - Run the script in the terminal using:

```
        python program.py
```

  - Provide inputs as prompted (e.g., Department ID or Product ID).

Program 1 Code: List Products or Departments

```python
import warnings
warnings.filterwarnings(action='ignore', module='.*paramiko.*')

from sshtunnel import SSHTunnelForwarder
import pymysql

def list_products_or_departments():
    try:
        print("Establishing SSH tunnel...")
        with SSHTunnelForwarder(
            ('fries.it.uu.se', 22),  # SSH server hostname and port
            ssh_username='Your Username',  # SSH username
            ssh_password='Your Password',  # SSH password
            remote_bind_address=('fries.it.uu.se', 3306)  # MySQL server hostname and port
        ) as tunnel:
            print(f"SSH tunnel established successfully! Local port: {tunnel.local_bind_port}")
            print("Connecting to the database...")

            conn = pymysql.connect(
                host='127.0.0.1',  # Always localhost when using the tunnel
                user='ht24_2_group_7',  # MySQL username
                password='pasSWd_7',  # MySQL password
                port=tunnel.local_bind_port,  # Tunnel local port
                database='ht24_2_project_group_7'  # Database name
            )
            print("Database connection established successfully!")

            cursor = conn.cursor()

            department_id = input("\nEnter Department ID: ").strip()

            query_check_child = """
            SELECT COUNT(*) AS child_count
            FROM Department
            WHERE Parent_Dept_ID = %s
            """
            cursor.execute(query_check_child, (department_id,))
            result = cursor.fetchone()

            if result and result[0] > 0:
                query_child_departments = """
                SELECT Department_ID, Title
                FROM Department
                WHERE Parent_Dept_ID = %s
                """
                cursor.execute(query_child_departments, (department_id,))
                rows = cursor.fetchall()

                if rows:
                    print("\nChild Departments:")
                    for row in rows:
                        print(f"ID: {row[0]}, Title: {row[1]}")
                else:
                    print("\nNo child departments found.")
            else:
                print("\nNo child departments found.")
                query_products = """
                SELECT ProductID, Title,
                       ROUND(PriceWithoutVAT * (1 + VATPercentage/100) * (1 - DiscountPercentage/100),2) AS RetailPrice
                FROM Product
                WHERE Department_ID = %s
                """
                cursor.execute(query_products, (department_id,))
                rows = cursor.fetchall()

                if rows:
                    print("\nProducts in Department:")
                    for row in rows:
                        print(f"ID: {row[0]}, Title: {row[1]}, Retail Price: {row[2]}")
                else:
                    print("\nNo products found in this department.")

    except pymysql.MySQLError as db_err:
        print(f"Database error: {db_err}")
    except Exception as err:
        print(f"An unexpected error occurred: {err}")
    finally:
        try:
            if 'conn' in locals():
```

```python
                cursor.close()
                conn.close()
                print()
                print("Database connection closed.")
        except Exception as cleanup_err:
            print(f"Error during cleanup: {cleanup_err}")
        print("SSH tunnel closed.")

if __name__ == "__main__":
    list_products_or_departments()
```

Program 2 Code: Update Product Discount

```python
import warnings
warnings.filterwarnings(action='ignore', module='.*paramiko.*')

from sshtunnel import SSHTunnelForwarder
import pymysql

def update_product_discount():
    try:
        print("Establishing SSH tunnel...")
        with SSHTunnelForwarder(
            ('fries.it.uu.se', 22),  # SSH server hostname and port
            ssh_username='Your Username',  # SSH username
            ssh_password='Your Password',  # SSH password
            remote_bind_address=('fries.it.uu.se', 3306)  # MySQL server hostname and port
        ) as tunnel:
            print(f"SSH tunnel established successfully! Local port: {tunnel.local_bind_port}")
            print("Connecting to the database...")

            conn = pymysql.connect(
                host='127.0.0.1',  # Always localhost when using the tunnel
                user='ht24_2_group_7',  # MySQL username
                password='pasSWd_7',  # MySQL password
                port=tunnel.local_bind_port,  # Tunnel local port
                database='ht24_2_project_group_7'  # Database name
            )
            print("Database connection established successfully!")

            cursor = conn.cursor()
            print()
            product_id = input("Enter Product ID: ").strip()

            query_current_discount = """
            SELECT DiscountPercentage
            FROM Product
            WHERE ProductID = %s
            """
            cursor.execute(query_current_discount, (product_id,))
            result = cursor.fetchone()

            if result:
                print(f"Current Discount: {result[0]}%")
                new_discount = float(input("Enter New Discount Percentage: "))

                query_update_discount = """
                UPDATE Product
                SET DiscountPercentage = %s
                WHERE ProductID = %s
                """
                cursor.execute(query_update_discount, (new_discount, product_id))
                conn.commit()

                print("Discount updated successfully!")
            else:
                print("Product not found.")

    except pymysql.MySQLError as db_err:
        print(f"Database error: {db_err}")
    except Exception as err:
        print(f"An unexpected error occurred: {err}")
    finally:
        try:
            if 'conn' in locals():
                cursor.close()
                conn.close()
                print()
                print("Database connection closed.")
        except Exception as cleanup_err:
            print(f"Error during cleanup: {cleanup_err}")
        print("SSH tunnel closed.")

if __name__ == "__main__":
    update_product_discount()
```