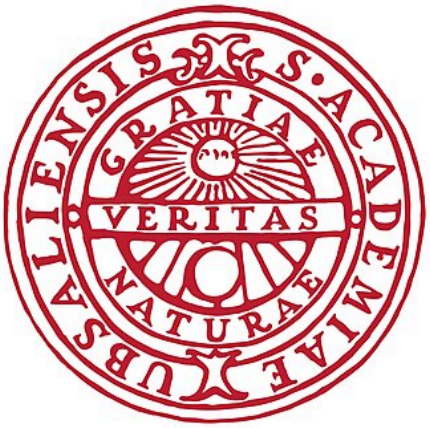


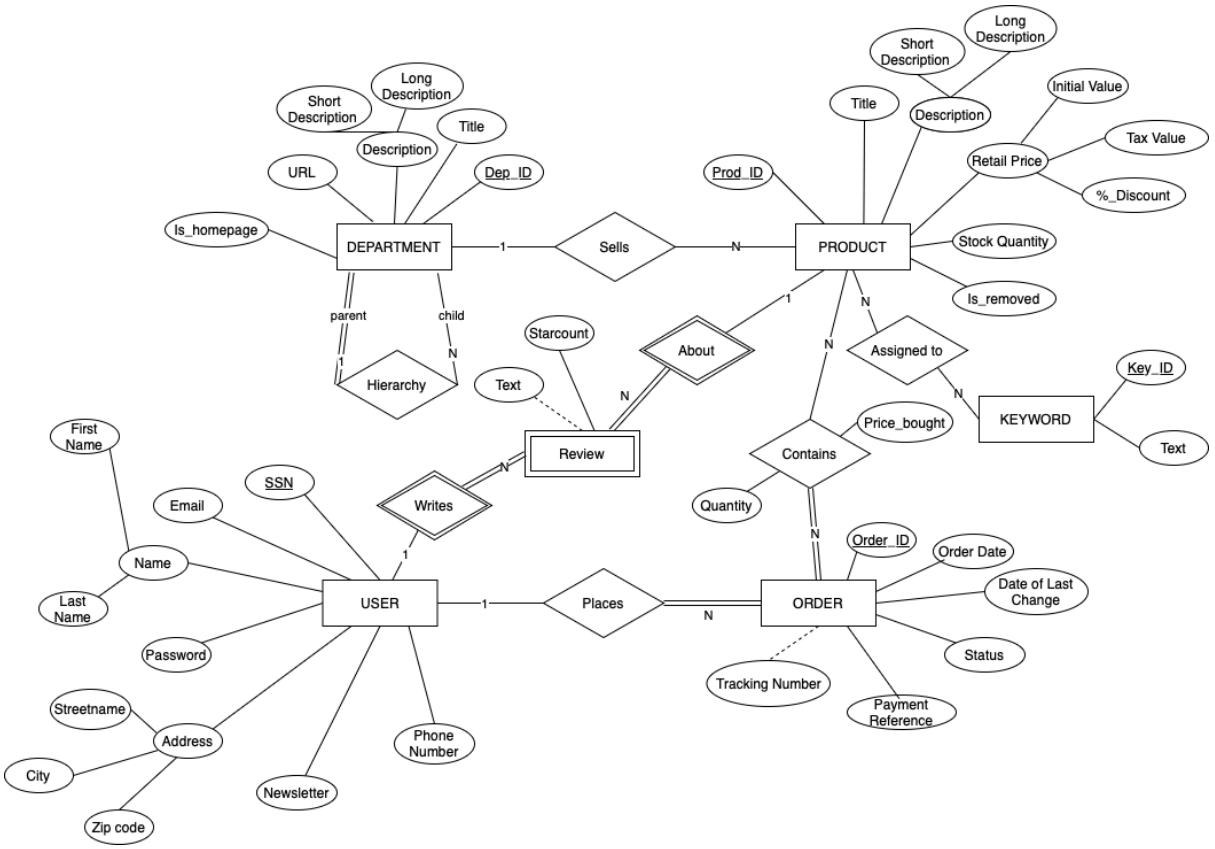
UPPSALA UNIVERSITET

Databases I Project

Project Group 24

Name:	Email Address:
Timothy Leow	timothy_leow@u.nus.edu
Jordi Catafal	jordi.catafal@estudiantat.upc.edu
Zhiping Hu	zhiping.hu.3994@student.uu.se
Patel Hetvi	patelhetvi1711@gmail.com

Task 1: EER-diagram



Above is our ER diagram for Milestone 1. Notable decisions made about the design of the database are explained below. In particular, we look into design decisions for Entities, and the Relations between them.

Entities

Department

- Dep_ID: Unique key.
- Is_homepage**:** A boolean attribute to indicate if the Department is the root Department that contains the welcome text as its description. Its value is True only for the root Department (homepage) and False for all other departments.
- URL: Contains the breadcrumbs: e.g, *"Home/Electronics/Computer and tablets/Tablets"*
- Description: Contains children attributes Long Description and Short Description which will be queried based on what the webpage needs to be displayed in the current page. (In the statement it's said that it needs a Description and a shorter one, so depending on what has to be used the frontend will query one or the other.)
- LOGO: We consider the Logo as a front-end matter.

Product

- Prod_ID: Unique key.

- Is_removed**:** A boolean attribute that indicates if a product is removed from the store. But we want to keep it in the database in order to not lose information.
- Description: Contains children attributes Long Description and Short Description which will be queried based on what the webpage needs to be displayed in the current page. (In the statement it's said that it needs a Description and a shorter one, so depending on what has to be used the frontend will query one or the other.)
- Retail Price: Contains relevant child attributes which will help to give information about taxes/discounts and the Product's original price.

Keyword

We decided to model it as a separate entity, to make querying keyword-related products easier, as compared to if Keyword is a multi-valued attribute of product.

- Key_id: The primary key of a keyword.
- Text: The keyword. (Single word)

User

- SSN: Unique key.
- Name: Contains children attributes First Name and Last Name

Review

- Starcount: a certain number that is given by a registered user to rate a product,1 star being a really bad product and 5 stars being a really good product.
- Text: An optional attribute (as indicated by the dotted line). comments about the product made by the registers with long plain text.

Order

- Tracking Number: An optional attribute (as indicated by the dotted line). An order may or may not have a tracking number for traceability purposes.

Relations:

Department **Sells** Products

A Department can sell many products, but each product should only belong to one leaf department (not including parent departments via transitivity).

Department **Parent/Child**

A parent department can include several child departments, but each child department should only have 1 parent department.

Products **Assigned to** Keywords

A product may contain several keywords and a keyword may appear in tags of many products. So this is a M:N relation.

Order **Contains** Products

An order can contain multiple products, and a product can be part of multiple orders. The price and quantity at which the product was bought is contained as attributes of this relation.

User **Places** Orders

A user can place many orders, but an order can only be placed by one user.

User **Writes** Reviews

A user can write many reviews, but each review can only be written by one user.

Reviews **About** Product

A review is written about a specific product, but a product can have multiple reviews from different users.

Task 2: Normalization

- **1NF**: Ensure that all columns have atomic values, and there are no repeating groups or arrays.
- **2NF**: Eliminate partial dependencies, so every non-key column depends on the **entire** primary key.
- **3NF**: Eliminate transitive dependencies, so non-key columns only depend on the **primary key**.

UNF:

DEPARTMENT	**PRODUCT**
Dep_ID Title Short_Description Long_Description URL Is_homepage Parent_Dep_ID (FK referencing DEPARTMENT)	Prod_ID Title Short_Description Long_Description Retail_Price_With_Taxes Tax_Value %_Discount Stock_Quantity Is_removed Dep_ID (FK referencing DEPARTMENT)
USER	**ORDER**
SNN First_Name Last_Name Email Password Street_name City Zip_code Phone_Number Newsletter	Order_ID Order_Date Date_of_Last_Change Status Tracking_Number Payment_Reference SSN (FK referencing USER)
REVIEW	**KEYWORD**
Review_ID Text Starcount SSN (FK referencing USER) Prod_ID (FK referencing PRODUCT)	Key_ID Text
ORDER_PRODUCT (“Contains”)	**PRODUCT_KEYWORD (“Assigned to”)**
Order_ID (FK referencing ORDER) Prod_ID (FK referencing PRODUCT) Quantity	Prod_ID (FK referencing PRODUCT) Key_ID (FK referencing KEYWORD)

Price_bought	

1NF:

The tables are already in 1NF as all attributes contain atomic values and there are no repeating groups.

2NF:

To achieve 2NF, we need to remove partial dependencies

DEPARTMENT: Already in 2NF (Dep_ID is the primary key)
PRODUCT: Already in 2NF (Prod_ID is the primary key)
USER: Already in 2NF (SSN is the primary key)
ORDER: Already in 2NF (Order_ID is the primary key)
REVIEW: Already in 2NF (Review_ID is the primary key)
KEYWORD: Already in 2NF (Key_ID is the primary key)
ORDER_PRODUCT: Already in 2NF (Order_ID, Prod_ID is the composite primary key)
PRODUCT_KEYWORD: Already in 2NF (Prod_ID, Key_ID is the composite primary key)

3NF:

To achieve 3NF, we need to remove transitive dependencies.

DEPARTMENT: No apparent transitive dependencies
USER: No apparent transitive dependencies
ORDER: No apparent transitive dependencies
REVIEW: No apparent transitive dependencies
KEYWORD: No apparent transitive dependencies
ORDER_PRODUCT: No apparent transitive dependencies
PRODUCT_KEYWORD: No apparent transitive dependencies

PRODUCT	**PRODUCT_PRICE**
Prod_ID Title Short_Description Long_Description URL Stock_Quantity Is_removed Dep_ID (FK referencing DEPARTMENT)	Prod_ID (FK referencing PRODUCT) Retail_Price_With_Taxes Tax_Value %_Discount

Task 3: Create Tables

```
CREATE TABLE DEPARTMENT (
  Dep_ID INT PRIMARY KEY,
  Title VARCHAR(255) NOT NULL,
  Short_Description TEXT,
  Long_Description TEXT,
  URL VARCHAR(255),
  Is_homepage BOOLEAN NOT NULL,
  Parent_Dep_ID INT,
  FOREIGN KEY (Parent_Dep_ID) REFERENCES DEPARTMENT(Dep_ID)
);
```

```
CREATE TABLE PRODUCT (
  Prod_ID INT PRIMARY KEY,
  Title VARCHAR(255) NOT NULL,
  Is_Featured TINYINT(1) NOT NULL,
  Short_Description TEXT,
  Long_Description TEXT,
  Stock_Quantity INT NOT NULL,
  Is_removed BOOLEAN NOT NULL,
  Dep_ID INT,
  FOREIGN KEY (Dep_ID) REFERENCES DEPARTMENT(Dep_ID)
);
```

```
CREATE TABLE USER (
  SSN VARCHAR(20) PRIMARY KEY,
  First_Name VARCHAR(50) NOT NULL,
  Last_Name VARCHAR(50) NOT NULL,
  Email VARCHAR(255) NOT NULL,
  Password VARCHAR(255) NOT NULL,
  Street_name VARCHAR(255),
  City VARCHAR(100),
  Zip_code VARCHAR(20),
  Phone_Number VARCHAR(20),
  Newsletter BOOLEAN
);
```

```
CREATE TABLE PRODUCT_PRICE (
  Prod_ID INT PRIMARY KEY,
  Retail_Price_With_Taxes DECIMAL(10, 2) NOT NULL,
  Tax_Value DECIMAL(5, 2) NOT NULL,
  Discount_Percentage DECIMAL(5, 2) NOT NULL,
```

```
  FOREIGN KEY (Prod_ID) REFERENCES PRODUCT(Prod_ID)
);
```

```
CREATE TABLE `ORDER` (
  Order_ID INT PRIMARY KEY,
  Order_Date DATE NOT NULL,
  Date_of_Last_Change DATE,
  Status VARCHAR(50) NOT NULL,
  Tracking_Number VARCHAR(50),
  Payment_Reference VARCHAR(50),
  SSN VARCHAR(20),
  FOREIGN KEY (SSN) REFERENCES USER(SSN)
);
```

```
CREATE TABLE REVIEW (
  Review_ID INT PRIMARY KEY,
  Text TEXT,
  Starcount INT NOT NULL,
  SSN VARCHAR(20),
  Prod_ID INT,
  FOREIGN KEY (SSN) REFERENCES USER(SSN),
  FOREIGN KEY (Prod_ID) REFERENCES PRODUCT(Prod_ID)
);
```

```
CREATE TABLE KEYWORD (
  Key_ID INT PRIMARY KEY,
  Text VARCHAR(50) NOT NULL
);
```

```
CREATE TABLE ORDER_PRODUCT (
  Order_ID INT,
  Prod_ID INT,
  Quantity INT NOT NULL,
  Price_bought DECIMAL(10, 2) NOT NULL,
  PRIMARY KEY (Order_ID, Prod_ID),
  FOREIGN KEY (Order_ID) REFERENCES `ORDER`(Order_ID),
  FOREIGN KEY (Prod_ID) REFERENCES PRODUCT(Prod_ID)
);
```

```
CREATE TABLE PRODUCT_KEYWORD (
  Prod_ID INT,
  Key_ID INT,
  PRIMARY KEY (Prod_ID, Key_ID),
```

```
  FOREIGN KEY (Prod_ID) REFERENCES PRODUCT(Prod_ID),
  FOREIGN KEY (Key_ID) REFERENCES KEYWORD(Key_ID)
);
```

Task 4: Populate tables

SQL to create departments: (1 home, 2 top-level, 3 child departments each)

INSERT INTO DEPARTMENT (Dep_ID, Title, Short_Description, Long_Description, URL, Is_homepage)
VALUES (1, 'Home', 'Home goods', 'Welcome to the department! Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam ipsum lectus, ultrices at est ut, interdum commodo justo. Vestibulum vel.', '/home', TRUE);

INSERT INTO DEPARTMENT (Dep_ID, Title, Short_Description, Long_Description, URL, Is_homepage, Parent_Dep_ID)
VALUES
(2, 'Electronics', 'For electronic needs', 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam ipsum lectus, ultrices at est ut, interdum commodo justo. Vestibulum vel.', '/home/electronics', FALSE, 1),
(3, 'Mobile Phones', 'For daily use', 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam ipsum lectus, ultrices at est ut, interdum commodo justo. Vestibulum vel.', '/home/electronics/mobile_phones', FALSE, 2),
(4, 'Laptops', 'For school or work', 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam ipsum lectus, ultrices at est ut, interdum commodo justo. Vestibulum vel.', '/home/electronics/laptops', FALSE, 2),
(5, 'Cameras', 'For photography', 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam ipsum lectus, ultrices at est ut, interdum commodo justo. Vestibulum vel.', '/home/electronics/cameras', FALSE, 2),
(6, 'Appliances', 'For homely needs', 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam ipsum lectus, ultrices at est ut, interdum commodo justo. Vestibulum vel.', '/home/appliances', FALSE, 1),
(7, 'Refrigerators', 'For cooling needs', 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam ipsum lectus, ultrices at est ut, interdum commodo justo. Vestibulum vel.', '/home/appliances/refrigerators', FALSE, 6),

(8, 'Washing Machines', 'For washing needs', 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam ipsum lectus, ultrices at est ut, interdum commodo justo. Vestibulum vel.', '/home/appliances/washing_machines', FALSE, 6),
(9, 'Microwaves', 'For heating needs', 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam ipsum lectus, ultrices at est ut, interdum commodo justo. Vestibulum vel.', '/home/appliances/microwaves', FALSE, 6);

SQL to create 10 products (1 to 2 products per leaf-level department)

INSERT INTO PRODUCT (Prod_ID, Title, Short_Description, Long_Description, Stock_Quantity, Is_removed, Dep_ID)
VALUES
(1, 'iPhone 15', '2023 product from Apple', 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam ipsum lectus, ultrices at est ut, interdum commodo justo. Vestibulum vel.', 200, FALSE, 3),
(2, 'iPhone 16', '2024 product from Apple', 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam ipsum lectus, ultrices at est ut, interdum commodo justo. Vestibulum vel.', 100, FALSE, 3),
(3, 'Windows XPS', 'Flagship laptop from Dell', 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam ipsum lectus, ultrices at est ut, interdum commodo justo. Vestibulum vel.', 200, FALSE, 4),
(4, 'Macbook Pro M4 Pro', 'Flagship laptop from Apple', 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam ipsum lectus, ultrices at est ut, interdum commodo justo. Vestibulum vel.', 100, FALSE, 4),
(5, 'Nikon DSLR', 'Flagship camera from Nikon', 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam ipsum lectus, ultrices at est ut, interdum commodo justo. Vestibulum vel.', 500, FALSE, 5),
(6, 'Canon DSLR', 'Flagship camera from Canon', 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam ipsum lectus, ultrices at est ut, interdum commodo justo. Vestibulum vel.', 500, FALSE, 5),
(7, 'Samsung Fridge', 'Flagship fridge from Samsung', 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam ipsum lectus, ultrices at est ut, interdum commodo justo. Vestibulum vel.', 500, FALSE, 7),
(8, 'Mitsubishi Fridge', 'Flagship fridge from Mitsubushi', 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam ipsum lectus, ultrices at est ut, interdum commodo justo. Vestibulum vel.', 500, FALSE, 7),
(9, 'Electrolux Washing Machine', 'Flagship washing machine from Electrolux', 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam ipsum lectus, ultrices at est ut, interdum commodo justo. Vestibulum vel.', 500, FALSE, 8),
(10, 'Electrolux Microwave', 'Flagship microwave from Electrolux', 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam ipsum lectus, ultrices at est ut, interdum commodo justo. Vestibulum vel.', 500, FALSE, 9);

SQL to create users

```
INSERT INTO USER (SSN, First_Name, Last_Name, Email, Password, Street_name, City,
Zip_code, Phone_Number, Newsletter)
VALUES
(1, 'John', 'Doe', 'johndoe@mail.com', 'asdf', 'John Street 32', 'Uppsala', '123 12', 0761231212,
FALSE),
(2, 'Bob', 'Tan', 'bobtan@mail.com', 'asdfff', 'Bob Street 44', 'Uppsala', '123 14', '0721231212',
FALSE);
```

SQL to create reviews

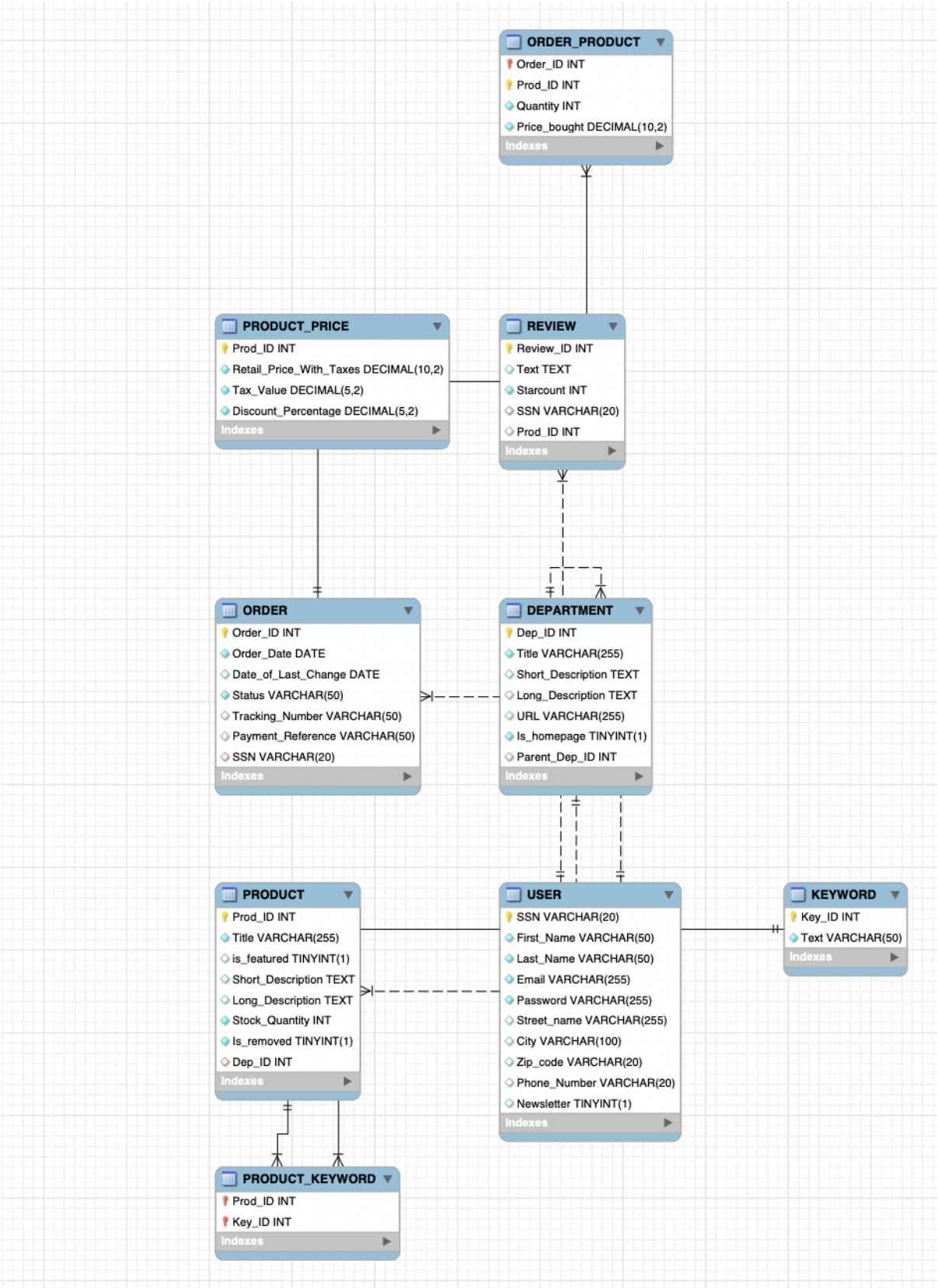
```
INSERT INTO REVIEW (Review_ID, Text, Starcount, SSN, Prod_ID)
VALUES
(1, "Great phone to use in 2023", 5, 1, 1),
(2, "Not the best phone to use in 2023", 4, 2, 1);
```

SQL to create order for one of the users (create order and order_product)

```
INSERT INTO `ORDER` (Order_ID, Order_Date, Status, Tracking_Number, Payment_Reference,
SSN)
VALUES
(1, '2024-10-07', 'Confirmed', '123456', '1', 1);

INSERT INTO ORDER_PRODUCT (Order_ID, Prod_ID, Quantity, Price_bought)
VALUES (1, 1, 5, 1500);
```

MySQL Workbench Diagram



Task 5: SQL Queries

• Welcome text for the homepage

```sql
SELECT Long_Description AS Welcome_Text
FROM DEPARTMENT
WHERE Is_homepage = 1;
```

100% 23:3

Result Grid Filter Rows: Search Export:

Welcome_Text
Welcome to our Homepage of Altonline!

• List of the top level departments with fields needed for the homepage

```sql
SELECT dep2.Title, dep2.Short_Description, dep2.URL
FROM DEPARTMENT dep1
JOIN DEPARTMENT dep2 ON dep1.Dep_ID = dep2.Parent_Dep_ID
WHERE dep1.Is_homepage = 1;
```

100% 30:4

Result Grid Filter Rows: Search Export:

Title	Short_Description	URL
Electronics	this is short description for a top-level Electronics department	127.0.0.1/home/electronics
Appliances	this is a short description for top-level Appliances department	127.0.0.1/home/Appliances

• List of the featured products with fields needed for the homepage

```sql
WITH RECURSIVE department_hierarchy AS (
    SELECT
        Dep_ID,
        Title,
        Parent_Dep_ID,
        Title AS full_path
    FROM DEPARTMENT
    WHERE Parent_Dep_ID IS NULL

    UNION ALL

    SELECT
        d.Dep_ID,
        d.Title,
        d.Parent_Dep_ID,
        CONCAT(dh.full_path, '/', d.Title) AS full_path
    FROM DEPARTMENT d
    INNER JOIN department_hierarchy dh ON d.Parent_Dep_ID = dh.Dep_ID
```

100% 41:1

Result Grid Filter Rows: Search Export:

Prod_ID	Title	Short_Description	product_link	retail_price
3	Google Pixel 9	Lorem ipsum dolor sit amet, consectetur adipisc...	Homepage/Electronics/Mobile Phones	10.00
8	Motorola	Lorem ipsum dolor sit amet, consectetur adipisc...	Homepage/Electronics/Mobile Phones	30.96
12	MacBook pro	Aliquam quam ligula, commodo et.	Homepage/Electronics/Laptops	21.60
15	Lenovo ThinkPad	Duis sit amet porta nunc. Ut sed diam. Duis sit...	Homepage/Electronics/Laptops	67.50
20	Galaxy Book	Quisque eget laoreet tortor. Cras posuere blandi...	Homepage/Electronics/Laptops	36.00
33	DJI Osmo Action Camera	Suspendisse tincidunt justo nisl, sed.	Homepage/Electronics/Cameras	98.01
63	Panasonic Inverter Microwave Oven	Suspendisse tincidunt justo nisl, sed.	Homepage/Appliances/Microwaves	56.70

• Given a product, list all keyword-related products

```sql
SELECT distinct p.Prod_ID, p.Title
FROM PRODUCT_KEYWORD pk1
JOIN PRODUCT_KEYWORD pk2 ON pk1.Key_ID = pk2.Key_ID
JOIN PRODUCT p ON pk2.Prod_ID = p.Prod_ID
WHERE pk1.Prod_ID = 5 and p.Prod_ID != pk1.Prod_ID
```

100% 51:5

Result Grid Filter Rows: Search Export:

Prod_ID	Title
15	Lenovo ThinkPad
35	Hasselblad X1D
65	Samsung Countertop Microwave Oven
73	Black+Decker Digital Microwave Oven

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

```sql
SELECT
    p.Prod_ID,
    p.Title,
    pp.Retail_Price_With_Taxes - pp.Tax_Value AS latest_price,
    AVG(r.starcount) AS Average_Rating
FROM PRODUCT p
JOIN PRODUCT_PRICE pp ON p.Prod_ID = pp.Prod_ID
LEFT JOIN REVIEW r ON p.Prod_ID = r.Prod_ID
WHERE p.Prod_ID = 35
GROUP BY p.Prod_ID, pp.Retail_Price_With_Taxes, pp.Tax_Value
```

100% 61:10

Result Grid Filter Rows: Search Export:

Prod_ID	Title	latest_price	Average_Rating
35	Hasselblad X1D	315.00	3.6667

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

```sql
SELECT p.Prod_ID, p.Title, pp.Discount_Percentage
FROM PRODUCT_PRICE pp
JOIN PRODUCT p ON pp.Prod_ID = p.Prod_ID
ORDER BY pp.Discount_Percentage DESC
```

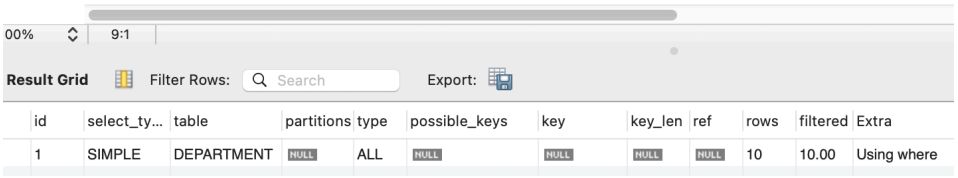
100% 37:4

Result Grid Filter Rows: Search Export:

Prod_ID	Title	Discount_Percenta...
15	Lenovo ThinkPad	50.00
33	DJI Osmo Action Cam...	33.00
12	MacBook pro	20.00
20	Galaxy Book	20.00
8	Motorola	18.00
35	Hasselblad X1D	15.00
3	Google Pixel 9	10.00
63	Panasonic Inverter Mi...	10.00

Task 6: Indexes

```
1 •  explain SELECT Long_Description AS Welcome_Text
2    FROM DEPARTMENT
3    WHERE Is_homepage = 1
```



00% 9:1

Result Grid Filter Rows: Search Export:

id	select_ty...	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	DEPARTMENT	NULL	ALL	NULL	NULL	NULL	NULL	10	10.00	Using where

This is the first query from task 5.

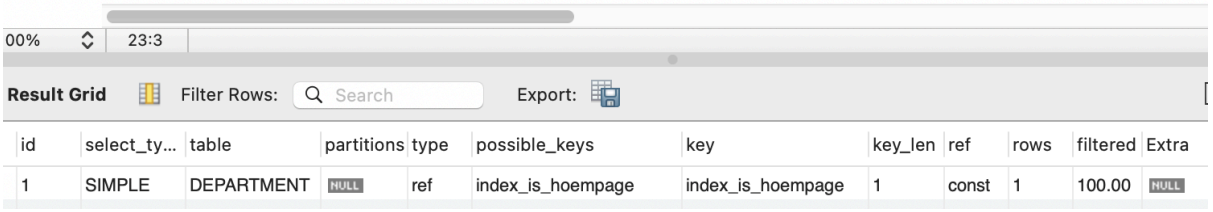
Here in order to select the welcome_text the primary key is used and it makes it kind of inefficient to access frequently, since this is the text that will appear on the homepage.

Is true that in our case we only have 10 rows, but in a real case scenario where there are a bunch of products we can have efficiency problems.

Due to the fact that we found this inefficiency, we created the first index using Btree.

CREATE INDEX idx_is_homepage_btree ON DEPARTMENT (Is_homepage) USING BTREE;

```
1 •  explain SELECT Long_Description AS Welcome_Text
2    FROM DEPARTMENT
3    WHERE Is_homepage = 1;
```



00% 23:3

Result Grid Filter Rows: Search Export:

id	select_ty...	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	DEPARTMENT	NULL	ref	index_is_hoempage	index_is_hoempage	1	const	1	100.00	NULL

After creating the index, as we can see on the image, our search engine has directly gone to the homepage instead of checking out all the departments that we have on our database.

Now we've done the same using a Hash Index:

CREATE INDEX idx_is_homepage_hash ON DEPARTMENT (Is_homepage) USING HASH;

But we found that the engine we were using does not support Hash indexes.

"***Warning | 3502 | This storage engine does not support the HASH index algorithm, storage engine default was used instead.***"

```
13 •  SHOW TABLE STATUS WHERE Name = 'DEPARTMENT';
14
15
```

100% 1:11

Result Grid Filter Rows: Search Export:

Name	Engine	Version	Row_format	Rows	Avg_row_length	Data_length	Max_data_leng...	Index_length	Data_free	Auto_increment	Create_time	Update_time
DEPARTMENT	InnoDB	10	Dynamic	10	1638	16384	0	16384	0	14	2024-10-09 10:00:35	2024-10-08 13:02:40

As we can see in the image we have here the engine for the Department table is "InnoDB" which only supports B Tree indexes.

Now we will do the same for the second query.

This is the query without the index

```
11 •  EXPLAIN SELECT dep2.Title, dep2.Short_Description, dep2.URL
12    FROM DEPARTMENT dep1
13    JOIN DEPARTMENT dep2 ON dep1.Dep_ID = dep2.Parent_Dep_ID
14    WHERE dep1.Is_homepage = 1;
```

100% 24:9

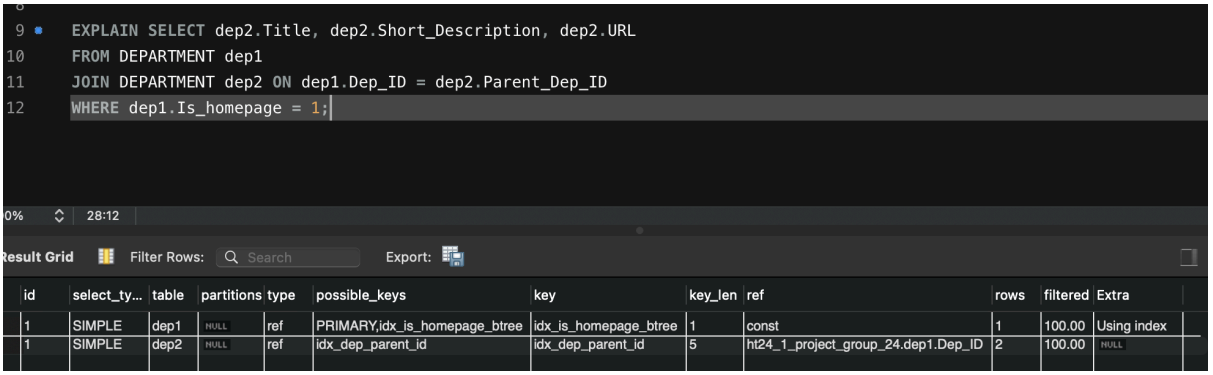
Result Grid Filter Rows: Search Export:

id	select_ty...	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	dep1	NULL	ALL	PRIMARY	NULL	NULL	NULL	10	10.00	Using where
1	SIMPLE	dep2	NULL	ref	idx_dep_parent_id	idx_dep_parent_id	5	ht24_1_project_group_24.dep1.Dep_ID	2	100.00	NULL

As we can see here, it goes through all the departments until it finds the one that it is searching for. So in order to avoid this inefficiency let's create indexes.

CREATE INDEX idx_is_homepage_btree ON DEPARTMENT (Is_homepage) USING BTREE;
CREATE INDEX idx_dep_parent_id ON DEPARTMENT (Parent_Dep_ID) USING BTREE;

After the index is created:



```
EXPLAIN SELECT dep2.Title, dep2.Short_Description, dep2.URL
FROM DEPARTMENT dep1
JOIN DEPARTMENT dep2 ON dep1.Dep_ID = dep2.Parent_Dep_ID
WHERE dep1.Is_homepage = 1;
```

0% 28:12

Result Grid Filter Rows: Search Export:

id	select_ty...	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	dep1	NULL	ref	PRIMARY,idx_is_homepage_btree	idx_is_homepage_btree	1	const	1	100.00	Using index
1	SIMPLE	dep2	NULL	ref	idx_dep_parent_id	idx_dep_parent_id	5	ht24_1_project_group_24.dep1.Dep_ID	2	100.00	NULL

We created 2 indexes since each one addresses a specific part of the query that can be optimized. So they collectively enhance the performance by optimizing filtering and join operations.

We tried again with Hash indexes

CREATE INDEX idx_dep_is_homepage_hash ON DEPARTMENT (Is_homepage) USING HASH;
CREATE INDEX idx_dep_parent_id_hash ON DEPARTMENT (Parent_Dep_ID) USING HASH;

But we found the same error as before:

"Warning | 3502 | This storage engine does not support the HASH index algorithm, storage engine default was used instead."

Even though not being able to create hashing indexes, we've searched the differences:

B-Tree indexes and Hash indexes differ primarily in their structure and the types of queries they optimize. B-Tree indexes are balanced tree structures that store keys in an ordered fashion, making them highly efficient for range queries (like BETWEEN, >=, and <=) and for retrieving ordered data. Since the keys are ordered, traversing the tree provides results in sequence. B-Tree indexes are more versatile, handling both exact lookups and range queries effectively. However, they consume more space and can be slower for simple exact lookups when compared to hashing, due to the tree traversal required.

Hash indexes, on the other hand, rely on a hashing function that maps keys directly to a specific location in a hash table. This makes them incredibly fast for exact match queries (e.g., =) because, in the best-case scenario, they provide direct access to the data in constant time. However, hash indexes are not suitable for range queries or ordering, as the keys are not stored in any particular order. They also require mechanisms to handle collisions, which occur when multiple keys map to the same hash value.

Hash indexes are generally more efficient in terms of space but less flexible than B-Trees, especially when query patterns are more complex than simple equality comparisons.

Task 7: Python with SQL

Program 1: List child departments/products given a Department ID

Source code: `dept_id_query.py`
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).

```python
import pymysql
from sshtunnel import SSHTunnelForwarder
import getpass

# Get SSH and database credentials from the user
ssh_username = input("Enter your SSH username: ")
ssh_password = getpass.getpass("Enter your SSH password: ")
db_user = input("Enter your MySQL username: ")
db_password = getpass.getpass("Enter your MySQL password: ")

# Set up SSH tunnel
tunnel = SSHTunnelForwarder(
    ('fries.it.uu.se', 22),  # SSH server details
    ssh_username=ssh_username,
    ssh_password=ssh_password,
    remote_bind_address=('127.0.0.1', 3306)
)

# Start SSH tunnel
tunnel.start()

# Connect to MySQL database through the tunnel
mydb = pymysql.connect(
    host='127.0.0.1',
    user=db_user,
    password=db_password,
    port=tunnel.local_bind_port,
    db='ht24_1_project_group_24'
)
```

This is the function for connecting to MySQL databases through an SSH tunnel with username and password. using the input() and getpass.getpass() functions to securely capture the passwords without echoing them in the console. After starting the SSH tunnel, it connects to the MySQL database (ht24_1_project_group_24).

```python
# Function to check if the department is a leaf (has no child departments)
def is_leaf_department(cursor, dep_id):
    query = "SELECT Dep_ID FROM DEPARTMENT WHERE Parent_Dep_ID = %s"
    cursor.execute(query, (dep_id,))
    result = cursor.fetchall()
    return len(result) == 0  # Returns True if no child departments

# Function to list products in the department if it's a leaf
def list_products(cursor, dep_id):
    query = """
    SELECT P.Prod_ID, P.Title, PP.Retail_Price_With_Taxes - (PP.Retail_Price_With_Taxes * PP.Discount_Percentage / 100) AS Price_After_Discount
    FROM PRODUCT P
    JOIN PRODUCT_PRICE PP ON P.Prod_ID = PP.Prod_ID
    WHERE P.Dep_ID = %s AND P.Is_removed = FALSE
    """
    cursor.execute(query, (dep_id,))
    products = cursor.fetchall()

    print(f"Products in Department {dep_id}:")
    if products:
        for product in products:
            print(f"Product ID: {product[0]}, Title: {product[1]}, Price After Discount: ${product[2]:.2f}")
    else:
        print("No products found in this department.")
```

The first function, `is_leaf_department`, checks whether a specified department is a leaf department, which means it has no child departments. The function constructs an SQL query that selects `Dep_ID` from the `DEPARTMENT` table where the `Parent_Dep_ID` matches the provided `dep_id`. After executing the query, it fetches all results. If the length of the result set is zero, it indicates that the department does not have any child departments, and the function returns `True`. Otherwise, it returns `False`, indicating that the department is not a leaf.

```python
# Function to list child departments if it's not a leaf
def list_child_departments(cursor, dep_id):
    query = "SELECT Dep_ID, Title FROM DEPARTMENT WHERE Parent_Dep_ID = %s"
    cursor.execute(query, (dep_id,))
    child_departments = cursor.fetchall()

    print(f"Child Departments of Department {dep_id}:")
    if child_departments:
        for dept in child_departments:
            print(f"Department ID: {dept[0]}, Title: {dept[1]}")
    else:
        print("No child departments found.")
```

The `list_child_departments` function is designed to retrieve and display the child departments of a specified parent department identified by its `dep_id`. The function constructs an SQL query that selects the `Dep_ID` and `Title` from the `DEPARTMENT` table, filtering the results where the `Parent_Dep_ID` matches the provided `dep_id`. After executing the query, it fetches all matching child departments. The function then prints a header indicating which department's child departments are being listed. If child departments

exist, it iterates through the results and prints each child department's ID and title in a formatted manner. If no child departments are found, it outputs a message indicating that there are no child departments available for the specified parent department.

```python
# Main program logic
def main():
    mycursor = mydb.cursor()

    try:
        # Ask the user for a department ID
        dep_id = input("Enter Department ID: ")

        # Check if the department is a leaf
        if is_leaf_department(mycursor, dep_id):
            # If it's a leaf, list products
            list_products(mycursor, dep_id)
        else:
            # If it's not a leaf, list child departments
            list_child_departments(mycursor, dep_id)

    finally:
        # Close the MySQL cursor and the database connection
        mycursor.close()
        mydb.close()
        # Stop the SSH tunnel
        tunnel.stop()

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

The `main()` function serves as the central logic of the program, which manages user interaction and database operations. First, it creates a cursor object, `mycursor`, to facilitate SQL queries on the database connection `mydb`. It prompts the user to enter a department ID and checks whether the specified department is a leaf department (i.e., one without child departments) by calling the `is_leaf_department()` function. Depending on the result, it either lists the products in that department using `list_products()` or lists its child departments with `list_child_departments()`.

OUTPUT

```
PS C:\Users\Hp> python -u "c:\Users\Hp\Desktop\database design\main.py"
Enter your SSH password:
Enter your MySQL username: ht24_1_group_24
Enter your MySQL password:
Enter Department ID: 1
Child Departments of Department 1:
Department ID: 5, Title: Mobile Phones
Department ID: 6, Title: Laptops
Department ID: 7, Title: Cameras
PS C:\Users\Hp> python -u "c:\Users\Hp\Desktop\database design\main.py"
Enter your SSH username: hepa1825
Enter your SSH password:
Enter your MySQL username: ht24_1_group_24
Enter your MySQL password:
```

Program 2: List discount and allow user to modify it given a Product ID

Source code: `product_id_query.py`

- Firstly, we get the product ID as an input from the user, validating that it is a valid ID based on the actual number of products (we use the table PRODUCT_PRICE in this case)

```
prod_id_to_query = input("Enter the product ID that you wish to know the discount or change it: ")
mycursor.execute("SELECT COUNT(*) FROM PRODUCT_PRICE")
table_size = mycursor.fetchone()[0]
if not validate_prod_id(prod_id_to_query, mycursor, table_size):
    print(f"Invalid product ID. Please enter a valid product ID. There are only {table_size} products in the database.")
    return 1
```

```
def validate_prod_id(prod_id, mycursor, table_size):
    try:
        prod_id = int(prod_id)
        return 1 <= prod_id <= table_size
    except ValueError:
        return False
```

- Then, we get the discount of the product based on its ID, returning an error code if no product price is found

```
mycursor.execute(f"SELECT Discount_Percentage FROM PRODUCT_PRICE WHERE Prod_ID = {prod_id_to_query}")
original_discount = mycursor.fetchone()
if original_discount:
    original_discount = original_discount[0]  # Extract the actual value
    print(f"The original discount for product ID {prod_id_to_query} is {original_discount}%.")
else:
    print(f"Error: No price/discount found for product ID {prod_id_to_query}")
    return 2
```

- The final step is to ask the user if they want to change the discount.

```
response = input("Do you want to change the discount for this product? (y/n): ")
if response.lower() == 'y':
    new_discount = get_discount_from_user()
    update_discount(mycursor, prod_id_to_query, new_discount)
    mydb.commit()
    print(f"The original discount for product ID {prod_id_to_query} was {original_discount}%. The new discount is now set to {new_discount}%.")
else:
    print("The discount for product ID " + str(prod_id_to_query) + " remains as " + str(original_discount) + "%.")

print("Alright, we're done! Goodbye.")
mydb.close()
tunnel.stop()
```

- If the user agrees, `get_discount_from_user` and `update_discount` is triggered

```python
def get_discount_from_user():
    new_discount = 0
    while True:
        try:
            new_discount = float(input("Enter the new discount percentage: "))
            if 0 <= new_discount <= 100:
                break
            else:
                print("Please enter a valid discount percentage between 0 and 100.")
        except ValueError:
            print("Invalid input. Please enter a numeric value.")
    return new_discount
```

```python
def update_discount(mycursor, prod_id, new_discount):
    mycursor.execute(f"UPDATE PRODUCT_PRICE SET Discount_Percentage = {new_discount} WHERE Prod_ID = {prod_id}")
    print("The new discount has been set as " + str(new_discount) + "%, for product ID " + str(prod_id) + ".")
```

- Otherwise, the original discount remains and the program terminates.

Sample run:

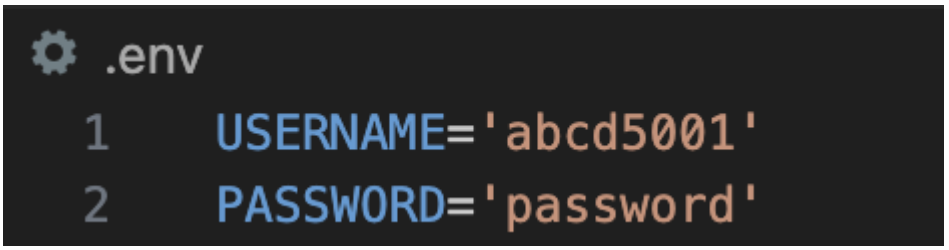
```
(.venv) (.venv) (base) timothyleow@timleows-Macbook-Pro Databases % python product_id_query.py
Enter the product ID that you wish to know the discount or change it: 63
The original discount for product ID 63 is 15.00%.
Do you want to change the discount for this product? (y/n): y
Enter the new discount percentage: 5
The new discount has been set as 5.0%, for product ID 63.
The original discount for product ID 63 was 15.00%. The new discount is now set to 5.0%.
Alright, we're done! Goodbye.
```

We can verify that the discount for product 63 is indeed now 5% by running a SELECT query: “SELECT Discount_Percentage FROM PRODUCT_PRICE WHERE Prod_ID=63

```
(.venv) (.venv) (base) timothyleow@timleows-Macbook-Pro Databases % python q.py
(Decimal('5.00'),)
```

Additional instructions:

The Studium username and password should be kept in a dotenv file like this:



pip packages you may need: pymysql, python-dotenv