Project in 1DL301 Database Design I

Group IT 13

Student1 `Nathaniel Ahy`

Student2 `Nils Hedberg`

Student3 `Joan Besora`

December 19, 2020

1 ER model

The ER diagram is depicted in Figure 1.

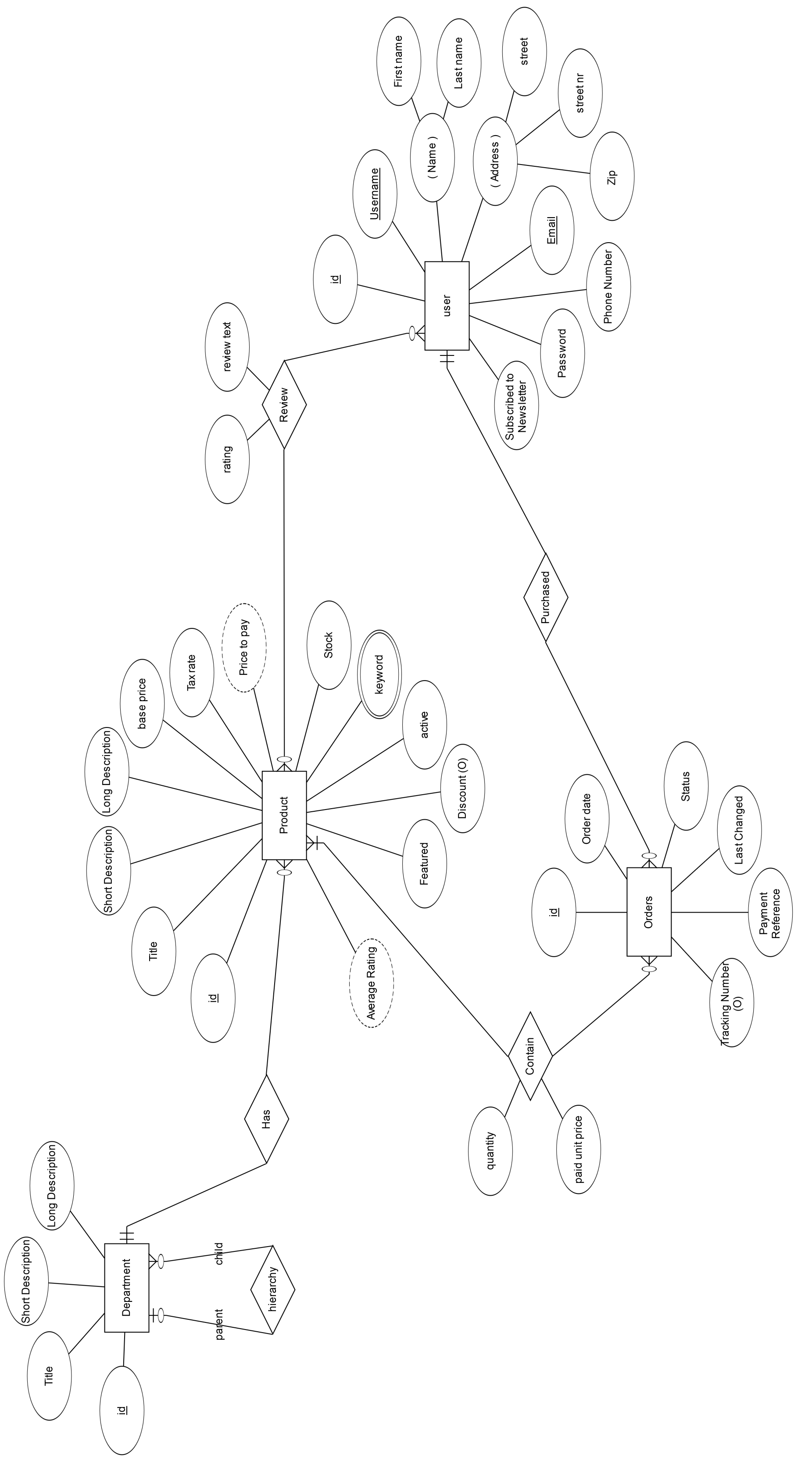


Figure 1: The ER diagram

2 SQL statements to create the tables

```sql
SET FOREIGN_KEY_CHECKS = 0;

DROP TABLE IF EXISTS Product;
DROP TABLE IF EXISTS Keyword;
DROP TABLE IF EXISTS Department;
DROP TABLE IF EXISTS User_Entity;
DROP TABLE IF EXISTS Orders;
DROP TABLE IF EXISTS Review;
DROP TABLE IF EXISTS Order_Content;


CREATE TABLE Product(
      id int NOT NULL PRIMARY KEY,
      title varchar(50) NOT NULL,
      short_description varchar(255),
      base_price decimal(10, 2),
      tax_rate decimal(10, 2),
      discount decimal(10, 2),
      active boolean,
      featured boolean,
      department_id int,
      stock int,
      long_description TEXT,
      FOREIGN KEY(department_id) REFERENCES Department(id)
);

CREATE TABLE Keyword(
      id int NOT NULL,
      product_id int NOT NULL,
      key_text varchar(50),
      PRIMARY KEY(id),
      FOREIGN KEY(product_id) REFERENCES Product(id)
);

CREATE TABLE Department(
      id int NOT NULL PRIMARY KEY,
      title varchar(50) NOT NULL,
      short_description varchar(255),
      long_description TEXT,
      parent_id int,
      FOREIGN KEY(parent_id) REFERENCES Department(id)
);

CREATE TABLE User_Entity(
      id int NOT NULL PRIMARY KEY,
      username varchar(50) NOT NULL UNIQUE,
      email varchar(50) NOT NULL UNIQUE,
      password varchar(50) NOT NULL,
      first_name varchar(50),
      last_name varchar(50),
      phone_number int,
      street varchar(50),
      street_nr int,
      zip int,
      subscribed_newsletter boolean
);

CREATE TABLE Orders(
      id int NOT NULL PRIMARY KEY,
      status varchar(50) NOT NULL,
      order_date date NOT NULL,
      last_changed date NOT NULL,
      payment_reference varchar(50),
      traking_number int,
      purchase_user int NOT NULL,
      FOREIGN KEY(purchase_user) REFERENCES User_Entity(id)
);

CREATE TABLE Review(
      product_id int NOT NULL,
      user_id int NOT NULL,
      rating int,
      review_text text,
      PRIMARY KEY(product_id, user_id),
      FOREIGN KEY(product_id) REFERENCES Product(id),
      FOREIGN KEY(user_id) REFERENCES User_Entity(id)
);

CREATE TABLE Order_Content(
      order_id int NOT NULL,
      product_id int NOT NULL,
      quantity int NOT NULL,
      paid_unit_price decimal(10, 2) NOT NULL,
      PRIMARY KEY(order_id, product_id),
      FOREIGN KEY(order_id) REFERENCES Orders(id),
      FOREIGN KEY(product_id) REFERENCES Product(id)
);

SET FOREIGN_KEY_CHECKS = 1;


-- Table population

INSERT INTO Department(id, title, short_description, long_description, parent_id)
VALUES
(10, 'Electronics', 'Electronics department', 'The electronics department', NULL),
(11, 'Phones', 'Phones department', 'The phones department', 10),
(12, 'Tablets', 'Tablets department', 'The tablets department', 10),
(20, 'Computers', 'Computers department', 'The computers department', NULL),
(21, 'Desktop computers', 'Desktop computer department',
'The desktop computer department', 20),
(22, 'Laptops', 'Laptops department', 'The laptops department', 20),
(100, 'Welcome', NULL, 'The welcome text of the webside', NULL);


INSERT INTO Product(id, title, short_description, base_price, tax_rate, discount,
active, featured, department_id, stock, long_description)
VALUES(20, 'Iphone 8', 'phone', 200, 10, 0, 1, 0, 11, 34, 'Very nice phone'),
(21, 'Iphone 7', 'phone', 100, 10, 0, 1, 0, 11, 27, 'Very nice phone'),
(22, 'Iphone 6', 'phone', 100, 10, 0, 0, 0, 11, 18, 'Very nice phone'),
(23, 'Ipad 8', 'tablet', 300, 10, 0, 0, 0, 12, 26, 'Very nice tablet'),
(24, 'Ipad 9', 'tablet', 400, 10, 20, 1, 0, 12, 31, 'Very nice tablet'),
(25, 'Ipad 10', 'tablet', 500, 10, 40, 1, 0, 12, 18, 'Very nice tablet'),
(26, 'HP desktop 3000', 'computer', 1000, 10, 0, 1, 1, 21, 33, 'Very nice computer'),
(27, 'HP desktop 5000', 'computer', 2000, 10, 0, 1, 0, 21, 29, 'Very nice computer'),
(28, 'HP laptop 2000', 'laptop', 3000, 10, 0, 1, 1, 22, 43, 'Very nice laptop'),
(29, 'HP laptop 4000', 'laptop', 5000, 10, 0, 1, 0, 22, 53, 'Very nice laptop');


INSERT INTO Keyword(id, product_id, key_text)
VALUES(30, 20, 'black'),
(31, 20, 'phone'),
(32, 20, 'expensive'),
(33, 21, 'white'),
(34, 21, 'phone'),
(35, 21, 'cheap'),
(36, 22, 'black'),
(37, 22, 'phone'),
(38, 22, 'cheap'),
(39, 23, 'tablet');


INSERT INTO User_Entity(id, username, email, password, first_name, last_name,
phone_number, street, street_nr, zip, subscribed_newsletter)
VALUES(40, 'Jonhy', 'aaa@bb.cc', 'a1a1', 'John', 'Smith', 333222444,
'Big street', 45, 44333, 1),
(41, 'Tommy', 'bbb@cc.dd', 'a2a2', 'Tom', 'Roberts', 222777888,
'Small street', 79, 22111, 0);


INSERT INTO Review(product_id, user_id, rating, review_text)
VALUES(21, 40, 10, 'Very good phone'),
(21, 41, 0, 'Very bad phone');


INSERT INTO Orders(id, status, order_date, last_changed, payment_reference,
traking_number, purchase_user)
VALUES(50, 'Shiping', '2019-06-06', '2019-08-08', 'Big bank', 6677, 41);


INSERT INTO Order_Content(order_id, product_id, quantity, paid_unit_price)
VALUES(50, 23, 1, 310),
(50, 24, 4, 350),
(50, 25, 1, 450);
```

3 MySQL model

The diagram generated by MySQL Workbench's Reverse Engineer for our database is depicted in Figure 2.

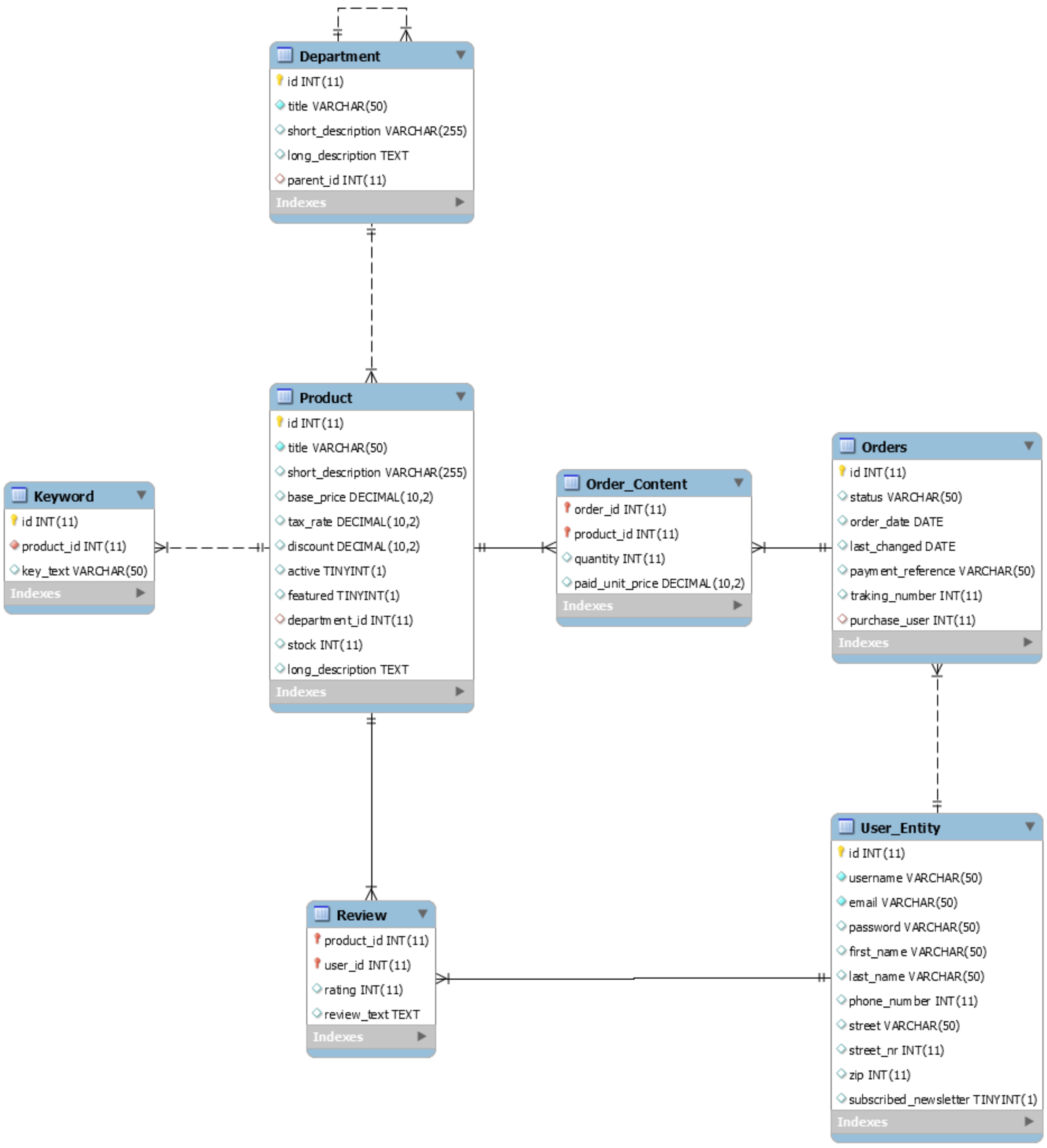


Figure 2: MySQL diagram

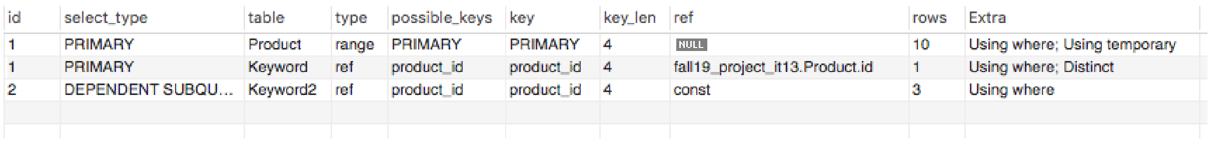
4 SQL Queries

```
-- 1. Select welcome text
SELECT long_description FROM Department
WHERE title='Welcome';

-- 2. Select top-level departments
SELECT title, short_description FROM Department
WHERE parent_id IS NULL AND title <>'Welcome';

-- 3. Select featured items for homepage
SELECT Title, Short_Description, base_price*(1-discount/100)*(1+tax_rate/100) AS\
```

```sql
retail_price
FROM Product
WHERE Featured = True AND active = True;

-- 4. Given a product with product id 20, list all related products
SELECT DISTINCT
Product.id, title, short_description,
(base_price * (1 - (discount/100)) * (1 + (tax_rate/100))) AS retail_price
FROM Product
LEFT JOIN Keyword ON Keyword.Product_ID = Product.ID
WHERE Product.ID != 20 AND EXISTS(
SELECT key_text, Product_ID
FROM Keyword AS Keyword2
WHERE Keyword.key_text = Keyword2.key_text AND Keyword2.Product_ID = 20
);

-- 5. Given department_id 2, select all products with specific fields
SELECT id, title, short_description,
(base_price * (1 - (discount/100)) * (1 + (tax_rate/100))) AS retail_price,
AVG(Review.rating) AS average_rating
FROM Product
LEFT JOIN Review ON Product.id = Review.product_id
WHERE department_id=2
GROUP BY Product.id;

-- 6. List all products on sale
SELECT *
FROM Product
WHERE active = 1 AND discount > 0
ORDER BY discount desc;
```

5 Query Optimization and Indices

Query 4 was selected for optimisation, before any indices were added performance looked like this:

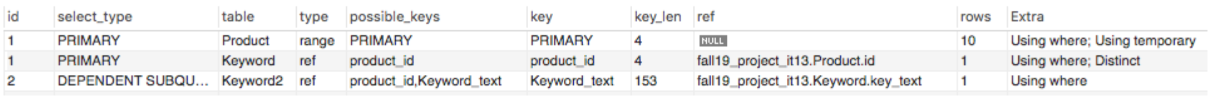


id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	Product	range	PRIMARY	PRIMARY	4	NULL	10	Using where; Using temporary
1	PRIMARY	Keyword	ref	product_id	product_id	4	fall19_project_it13.Product.id	1	Using where; Distinct
2	DEPENDENT SUBQU...	Keyword2	ref	product_id	product_id	4	const	3	Using where

After adding an index such as:

```sql
CREATE INDEX Keyword_text ON Keyword(key_text);
```

the performance improved as the number of rows are reduced.



id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	Product	range	PRIMARY	PRIMARY	4	NULL	10	Using where; Using temporary
1	PRIMARY	Keyword	ref	product_id	product_id	4	fall19_project_it13.Product.id	1	Using where; Distinct
2	DEPENDENT SUBQU...	Keyword2	ref	product_id,Keyword_text	Keyword_text	153	fall19_project_it13.Keyword.key_text	1	Using where

6 Task 6, Normalization, Dependencies, and Candidate keys

6.1 Full Functional Dependencies

Product There are no full-functional dependencies in Product aside from the dependencies the ID attribute entails. If we let {Product} denote the set of all attributes belonging to product. That is, the candidate key and only full functional dependency is depicted as:

$$\{ID\} \rightarrow \{Product\}$$

since 'ID' is unique as opposed to all other attributes belonging to product.

Users Since ID, Email, and Username are all unique as opposed to all other attributes they are the only candidate keys, therefore all values belonging to the User entity are fully functionally dependent on these attributes. If we let {User} denote all the attributes belonging to the User entity we can summarize the above as:

{ID}$\rightarrow \{User\}$
$\{Email\} \rightarrow \{User\}$
$\{Username\} \rightarrow \{User\}.$

Review The only candidate key and functional dependency is {User_Id,Product_Id} since you need both to know which user gave what rating and what review. More explicitly, if we denote {Review} as the set of all attributes belonging to the Review relation then the following relation holds:

$$\{User\_Id, Product\_Id\} \rightarrow \{Review\}.$$

6.2 Normal Forms

Product All functional dependencies within the Product entity hinge on candidate and/or superkeys. Moreover, since the candidate key is a single attribute it has no proper subset and therefore non-prime attributes are not fully functional dependent on its subset. Finally, no composite attributes, multivalued attributes, nor nested relations occur and therefore 'Product' is at least 1NF which, in conjunction with the above arguments, shows that it is BCNF.

User The User entity has no composite attributes, multivalued attributes, nor nested relations, meaning it is at least 1NF. All of its candidate keys are single attributes, therefore they have no proper subsets meaning that non-prime attributes cannot belong to their proper subsets. Finally, since the only full functional dependencies come from the candidate keys and/or superkeys the User entity is BCNF.

Review Review has only one candidate key, it's primary key, and neither the Rating nor the Review_Text attributes can be accessed with a proper subset of the primary key since a Review is given by a user to a certain product. Obviously, we have no composite attributes, multivalued attributes, nor nested relations (1NF); which was the last ingredient necessary to verify that Review is BCNF since the above arguments show that full functional dependencies are created by superkeys.

7 Python Program

7.1 Program 1 - Departments

```
from __future__ import print_function
import MySQLdb
```

```
conn = MySQLdb.connect("back.db1.course.it.uu.se", "fall19_it13", "Vk7BT9tJ",\
"fall19_project_it13")
conn.autocommit(True)
cursor = conn.cursor()

department = input("Enter department ID: ")
depProductQuery = """
SELECT Product.ID, Title, base_price*(1-discount/100)*(1-tax_rate/100) AS retail_price
FROM Product
WHERE Product.department_id = %s
"""

cursor.execute(depProductQuery, [department])

if(cursor.rowcount > 0):
   print("Department products: ")
   for (ID,name, price) in cursor:
      print("ID: ", ID, "Name: ", name, " , price: ", price)

depChildQuery = """
SELECT Department.ID, Title
FROM Department
WHERE Department.parent_id = %s
"""
cursor.execute(depChildQuery, [department])

if(cursor.rowcount > 0):
   print("Subdepartments: ")
   for (ID, name) in cursor:
      print("ID: ", ID, "Name: ", name)

cursor.close()
conn.close()
```

7.2 Program 2 - Product Discount

```
from __future__ import print_function
import MySQLdb

conn = MySQLdb.connect("back.db1.course.it.uu.se", "fall19_it13", "Vk7BT9tJ",\
"fall19_project_it13")
conn.autocommit(True)
cursor = conn.cursor()

product = input("Enter a valid product ID: ")

productQuery = """
SELECT Discount
FROM Product
WHERE Product.ID = %s
"""

cursor.execute(productQuery, [product])

print("Product discount: ", cursor.fetchall()[0][0])
```

```
newDiscount = input("New discount value: ")

updateQuery = """
UPDATE Product
SET discount = %s
WHERE Product.ID = %s
"""

cursor.execute(updateQuery, [newDiscount, product])

cursor.close()
conn.close()
```