 ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ

**Катедра „Компютърни системи”**

**КУРСОВ ПРОЕКТ ПО БАЗИ ОТ ДАННИ**

# Студент: Иван Иванов Стоилов

**Фак. №: 121221122 Група: 40**

|  |
| --- |
| **Тема №30** |
| Да се разработи база данни за система за склад на търговска верига за битова техника, в който стоките са записани по групи, марки, модели, количества и цени. Потребителите да могат да заявяват дадени стоки и да получават определени предварително дефинирани отстъпки за по-големи количества. Допълнете таблиците с необходима информация по ваш избор. |

1. Да се проектира база от данни и да се представи ER диаграма със съответни CREATE TABLE заявки за средата MySQL.
2. Напишете заявка, в която демонстрирате SELECT с ограничаващо условие по избор.
3. Напишете заявка, в която използвате агрегатна функция и GROUP BY по ваш избор.
4. Напишете заявка, в която демонстрирате INNER JOIN по ваш избор.
5. Напишете заявка, в която демонстрирате OUTER JOIN по ваш избор.
6. Напишете заявка, в която демонстрирате вложен SELECT по ваш избор.
7. Напишете заявка, в която демонстрирате едновременно JOIN и агрегатна функция.
8. Създайте тригер по ваш избор.
9. Създайте процедура, в която демонстрирате използване на курсор.

***Вашата работа трябва да включва: задание, ER-диаграма, CREATE TABLE заявки, всички останали заявки, решения на задачите от 2 до 9 и резултатите от тях.***

**Задача 1.** Да се проектира база от данни и да се представи ER диаграма със съответни CREATE TABLE заявки за средата MySQL.

Основните обекти, за които трябва да съхраняваме информация, според заданието са: StockGroups, Brands, Models и Orders. Допълнително ще създадем още няколко таблици, необходими за завършване на базата от данни – Payments, Deliveries, Discount, Stock и Clients.

Първата ще отразява съответната група, към която може да се причисли дадена стока. В нея ще се съдържа информация за името на съответната група (Електроника, Домакински електроуреди и др.).

Във втората таблица ще съхраняваме името на съответните марки (Samsung, Bosch, AEG, Liebherr и т.н.). В таблицата Models ще съхраняваме името на модела, като той ще бъде свързан със съответната марка и група към които принадлежи.

В следващата таблица, която ще създадем - Orders, ще съхраняваме поръчаното количество от дадена стока и датата на поръчката, като всичко това ще бъде свързано и със съответния клиент и със таблицата за наличност на стоките.

Ще създадем и допълнителните таблици, споменати и засегнати по-горе, които са както следва:

* Payments – съдържа информация за дължимата сума, датата на плащане, както и номера на поръчката и клиента.
* Deliveries – съдържа информация за доставеното количество стока, датата на доставка, както и номера на поръчката и клиента.
* Discount – съдържа информация за процента отстъпка, който клиентите получават при поръчано по-голямо количество от дадена стока, минималното количество стока, която трябва да поръчат за да получат отстъпка, както и номера на съответната стока.
* Stock – съдържа информация за наличното количество от дадена стока, цената на съответната стока, както и номерата на групата към която принадлежи, марката и модела на стоката.
* Clients – съдържа информация за клиентите, които поръчват стоки от склада – име на фирмата, адрес за доставка, телефон и e-mail за контакт.

За проектирането на базата ще използваме модела ER-диаграма (Entity Relationship Diagram):

A picture containing text, screenshot, number, parallel

Description automatically generated

Заявките, с които създаваме базата данни и таблиците, са:

DROP DATABASE IF EXISTS warehouse;

CREATE DATABASE wаrehouse;

USE wаrehouse;

CREATE TABLE clients(

id int not null auto\_increment PRIMARY KEY,

name varchar(255) not null,

address varchar(255) not null,

phone varchar(12) not null,

email varchar(255)

);

CREATE TABLE stockGroups(

id int not null auto\_increment PRIMARY KEY,

name VARCHAR(255) NOT NULL

);

CREATE TABLE brands(

id int not null auto\_increment PRIMARY KEY,

name VARCHAR(255) NOT NULL

);

CREATE TABLE models(

id int not null auto\_increment PRIMARY KEY,

name VARCHAR(255) NOT NULL,

brand\_id int not null,

group\_id int not null,

FOREIGN KEY(brand\_id) references brands(id),

FOREIGN KEY(group\_id) references stockGroups(id)

);

CREATE TABLE stock(

id int not null auto\_increment PRIMARY KEY,

groups\_id int not null,

brand\_id int not null,

model\_id int not null,

quantity int not null,

price double not null,

FOREIGN KEY (groups\_id) REFERENCES stockGroups(id),

FOREIGN KEY (brand\_id) REFERENCES brands(id),

FOREIGN KEY (model\_id) REFERENCES models(id)

);

CREATE TABLE orders(

id int not null auto\_increment PRIMARY KEY,

client\_id int not null,

stock\_id int not null,

or\_qty int not null,

dateOfOrder DATE not null,

FOREIGN KEY (client\_id) REFERENCES clients(id),

FOREIGN KEY (stock\_id) references stock(id)

);

CREATE TABLE discount(

id int not null auto\_increment PRIMARY KEY,

stock\_id int not null,

min\_qty int not null,

disc\_percentage int not null,

FOREIGN KEY (stock\_id) references stock(id)

);

CREATE TABLE payments(

id int not null auto\_increment PRIMARY KEY,

order\_id int not null,

client\_id int not null,

total double not null,

pay\_Date DATE not null,

FOREIGN KEY (client\_id) REFERENCES clients(id),

FOREIGN KEY (order\_id) references orders(id)

);

CREATE TABLE deliveries(

id int not null auto\_increment PRIMARY KEY,

stock\_id int not null,

client\_id int not null,

del\_qty int not null,

del\_date DATE not null,

FOREIGN KEY (client\_id) REFERENCES clients(id),

FOREIGN KEY (stock\_id) references stock(id)

);

Добавяме и тестови данни в таблиците:

INSERT INTO clients (name, address, phone, email)

VALUES ('Technopolis', 'ul. Okolovrasten pat 265', '02 921 1111', 'orders@technopolis.bg'),

('TechMart', 'bul. Simeonovsko Shose 138', '02 405 1379', 'orders@techmart.bg'),

('Zora', 'ul. Profesor Stoyan Kirkovich 13', '02 887 9495', 'orders@zora.bg');

INSERT INTO stockGroups (name)

VALUES ('Electronics'),

('Home Appliances');

INSERT INTO brands (name)

VALUES ('Samsung'),

('Bosch');

INSERT INTO models (name, brand\_id, group\_id)

VALUES ('Galaxy S20', 1, 1),

('KGN36NLEA', 2, 2);

INSERT INTO stock (groups\_id, brand\_id, model\_id, quantity, price)

VALUES (1, 1, 1, 100, 799.99),

(2, 2, 2, 50, 1099.99);

INSERT INTO orders (client\_id, stock\_id, or\_qty, dateOfOrder)

VALUES (1, 1, 2, '2022-01-01'),

(2, 2, 5, '2022-01-20'),

(3, 2, 7, '2022-01-20');

INSERT INTO discount (stock\_id, min\_qty, disc\_percentage)

VALUES (1, 10, 10),

(2, 5, 10);

INSERT INTO payments (order\_id, client\_id, total, pay\_Date)

SELECT orders.id, orders.client\_id, stock.price \* orders.or\_qty, CURDATE()

FROM orders

INNER JOIN stock ON orders.stock\_id = stock.id;

INSERT INTO deliveries (client\_id, stock\_id, del\_qty, del\_date)

VALUES (1, 1, 1, '2022-01-10'),

(2, 2, 1, '2022-01-20');

**Задача 2.** Напишете заявка, в която демонстрирате SELECT с ограничаващо условие по избор – заявката ще бъде комбинация от **Задача 2** и **Задача 7**. Тя ще покаже всички заявени от клиенти продукти през м.Януари 2022г. с тяхната марка и модел, както и количество на продукта, дата на заявката, дължима сума и информация за клиента.

SELECT orders.id as order\_ID, clients.name as client\_name, clients.address as clientAddress, clients.phone as clientPhone,

stockgroups.name as StockGroup, CONCAT(brands.name, " ", models.name) as ProductName, orders.or\_qty as Qty,

orders.dateOfOrder, CONCAT(payments.total, " BGN") as total

FROM stock

JOIN orders ON stock.id=orders.stock\_id

JOIN clients ON orders.client\_id=clients.id

JOIN brands ON stock.brand\_id=brands.id

JOIN models ON stock.model\_id=models.id

JOIN stockgroups ON stock.groups\_id=stockgroups.id

JOIN payments ON orders.id=payments.order\_id

WHERE dateOfOrder BETWEEN '2022-01-01' AND '2022-01-31'

Резултатът след изпълнението на тази заявка е:

A screenshot of a phone number

Description automatically generated with low confidence

**Задача 3.** Напишете заявка, в която използвате агрегатна функция и GROUP BY по ваш избор – показва дължимата сума от всеки клиент, заявил стока във склада.

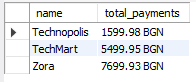
SELECT clients.name, CONCAT(SUM(payments.total), " BGN") AS total\_payments

FROM clients

JOIN payments ON payments.client\_id = clients.id

GROUP BY clients.name;

Резултатът от изпълнението на заявката е:



**Задача 4.** Напишете заявка, в която демонстрирате INNER JOIN по ваш избор - връща информация за датата на поръчката, артикула, който е поръчан, и магазина, който я е поръчал.

SELECT orders.id, clients.name, brands.name, models.name, orders.dateOfOrder

FROM clients

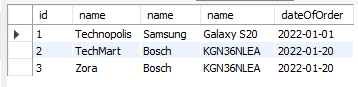
INNER JOIN orders ON clients.id = orders.client\_id

INNER JOIN stock ON orders.stock\_id = stock.id

INNER JOIN models ON stock.model\_id = models.id

INNER JOIN brands ON stock.brand\_id = brands.id;

Резултатът от изпълнението на заявката е:



**Задача 5.** Напишете заявка, в която демонстрирате OUTER JOIN по ваш избор – заявката ще ни даде списък на всички клиенти в базата данни, които имат поръчки, включително тези, които не са направили никакви поръчки, както и наличностите на заявената стока. Ако няма поръчки за дадения клиент, ще бъде изведена стойност null за колоната "quantity".

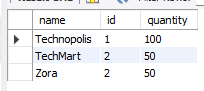
SELECT clients.name, stock.id, stock.quantity

FROM clients

LEFT OUTER JOIN orders ON clients.id = orders.client\_id

LEFT OUTER JOIN stock ON orders.stock\_id = stock.id;

Резултатът от изпълнението на заявката е:



**Задача 6.** Напишете заявка, в която демонстрирате вложен SELECT по ваш избор - ще изведе клиентско ID и средна стойност на плащанията за всеки клиент.

SELECT p.client\_id, AVG(p.total) AS average\_payment

FROM

(SELECT client\_id, SUM(total) AS total

FROM payments

GROUP BY client\_id) AS p

GROUP BY p.client\_id;

Резултатът от изпълнението на заявката е:

A screenshot of a computer

Description automatically generated with medium confidence

**Задача 7.** Напишете заявка, в която демонстрирате едновременно JOIN и агрегатна функция – както споменахме по-горе, тази задача е комбинирана със **задача 2** поради появилата се идея за заявка във тази задача и последвалата необходимост да се демонстрира едновременното използване на SELECT, JOIN и агрегатна функция.

**Задача 8.** Създайте тригер по ваш избор - за всеки нов запис в "orders", тригерът ще обнови количеството на продукта в таблицата "stock" чрез изваждане на количеството, посочено в новия запис от "orders".

DELIMITER |

DROP TRIGGER IF EXISTS update\_stock\_on\_order;

CREATE TRIGGER update\_stock\_on\_order AFTER INSERT ON orders

FOR EACH ROW

BEGIN

UPDATE stock

SET quantity = quantity - NEW.or\_qty

WHERE id = NEW.stock\_id;

END;

| DELIMITER ;

За да тестваме тригера пишем една допълнителна INSERT заявка, която ще промени наличността на артикул с номер 2.

INSERT INTO orders (client\_id, stock\_id, or\_qty, dateOfOrder)

VALUES (1, 2, 3, '2022-02-01');

От тестовите данни се вижда, че въведената наличност е 50бр. След изпълнението на тригера и заявката, за проверка пишем проста SELECT заявка:

SELECT quantity FROM stock WHERE id = 2;

Резултатът от нея е:

A screenshot of a computer

Description automatically generated with low confidence

**Задача 9.** Създайте процедура, в която демонстрирате използване на курсор .

Тази процедура приема име на клиент като входен параметър и използва курсор за да изведе информацията за всички поръчки на клиента заедно с информацията за моделите и марките на стоките, които са били поръчани.

DELIMITER |

DROP PROCEDURE IF EXISTS show\_client\_orders;

CREATE PROCEDURE show\_client\_orders(IN client\_name VARCHAR(255))

BEGIN

DECLARE total\_due DOUBLE;

DECLARE order\_id, client\_id, stock\_id, or\_qty, model\_id, brand\_id INT;

DECLARE model\_name, brand\_name VARCHAR(255);

DECLARE done INT DEFAULT FALSE;

DECLARE order\_cursor CURSOR FOR

SELECT o.id, o.client\_id, o.stock\_id, o.or\_qty, m.id, b.id

FROM orders o

INNER JOIN stock s ON o.stock\_id = s.id

INNER JOIN models m ON s.model\_id = m.id

INNER JOIN brands b ON m.brand\_id = b.id

INNER JOIN clients c ON o.client\_id = c.id

WHERE c.name = client\_name;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN order\_cursor;

SET total\_due = 0;

За да използваме процедурата, трябва да я извикаме по следния начин:  
Резултатът от изпълнението на процедурата е:

CALL show\_client\_orders('Technopolis');

SELECT concat('Orders for client: ', client\_name) as result;

FETCH order\_cursor INTO order\_id, client\_id, stock\_id, or\_qty, model\_id, brand\_id;

WHILE NOT done DO

SELECT m.name INTO model\_name FROM models m WHERE m.id = model\_id;

SELECT b.name INTO brand\_name FROM brands b WHERE b.id = brand\_id;

SELECT (s.price \* o.or\_qty) INTO total\_due FROM orders o INNER JOIN stock s ON o.stock\_id = s.id WHERE o.id = order\_id;

SELECT concat('Order: ', order\_id, ', Brand: ', brand\_name, ', Model: ', model\_name, ', Quantity: ', or\_qty, ', Total: ', total\_due, ' BGN') as result

GROUP BY order\_id;

SET total\_due = 0;

FETCH order\_cursor INTO order\_id, client\_id, stock\_id, or\_qty, model\_id, brand\_id;

END WHILE;

CLOSE order\_cursor;

END |

DELIMITER ;

A screenshot of a computer

Description automatically generated with medium confidence

Показва номера на клиента, марката и модела на стоката, поръчаното количество и дължимата сума от клиента.