SQL Exercises/The computer store

Contents

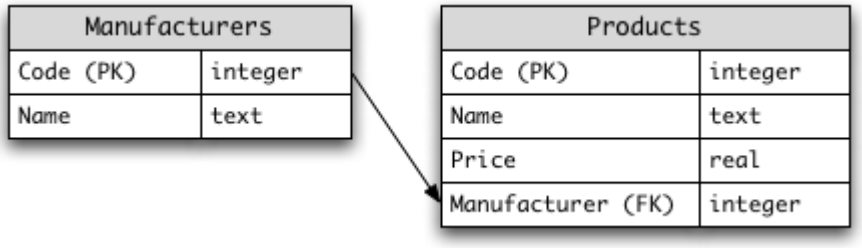
Relational Schema

Exercises

Table creation code

Sample dataset

Relational Schema



Please note the datatypes given are SQLite datatypes.

PK and FK stand for primary key and foreign key respectively.

Exercises

1. Select the names of all the products in the store.

Click to see solution

```
SELECT Name FROM Products;
```

2. Select the names and the prices of all the products in the store.

Click to see solution

```
SELECT Name, Price FROM Products;
```

3. Select the name of the products with a price less than or equal to $200.

Click to see solution

```
SELECT Name FROM Products WHERE Price <= 200;
```

4. Select all the products with a price between $60 and $120.

Click to see solution

```sql
/* With AND */
SELECT * FROM Products
  WHERE Price >= 60 AND Price <= 120;

/* With BETWEEN */
SELECT * FROM Products
  WHERE Price BETWEEN 60 AND 120;
```

5. Select the name and price in cents (i.e., the price must be multiplied by 100).

Click to see solution

```sql
/* Without AS */
SELECT Name, Price * 100 FROM Products;

/* With AS */
SELECT Name, Price * 100 AS PriceCents FROM Products;
```

6. Compute the average price of all the products.

Click to see solution

```sql
SELECT AVG(Price) FROM Products;
```

7. Compute the average price of all products with manufacturer code equal to 2.

Click to see solution

```sql
SELECT AVG(Price) FROM Products WHERE Manufacturer=2;
```

8. Compute the number of products with a price larger than or equal to $180.

Click to see solution

```sql
SELECT COUNT(*) FROM Products WHERE Price >= 180;
```

9. Select the name and price of all products with a price larger than or equal to $180, and sort first by price (in descending order), and then by name (in ascending order).

Click to see solution

```sql
   SELECT Name, Price
     FROM Products
    WHERE Price >= 180
ORDER BY Price DESC, Name;
```

10. Select all the data from the products, including all the data for each product's manufacturer.

Click to see solution

```sql
/* Without LEFT JOIN */
SELECT * FROM Products, Manufacturers
  WHERE Products.Manufacturer = Manufacturers.Code;

/* With LEFT JOIN */
SELECT *
  FROM Products LEFT JOIN Manufacturers
  ON Products.Manufacturer = Manufacturers.Code;
```

11. Select the product name, price, and manufacturer name of all the products.

Click to see solution

```sql
/* Without INNER JOIN */
SELECT Products.Name, Price, Manufacturers.Name
  FROM Products, Manufacturers
  WHERE Products.Manufacturer = Manufacturers.Code;

/* With INNER JOIN */
SELECT Products.Name, Price, Manufacturers.Name
  FROM Products INNER JOIN Manufacturers
  ON Products.Manufacturer = Manufacturers.Code;
```

12. Select the average price of each manufacturer's products, showing only the manufacturer's code.

Click to see solution

```sql
  SELECT AVG(Price), Manufacturer
    FROM Products
GROUP BY Manufacturer;
```

13. Select the average price of each manufacturer's products, showing the manufacturer's name.

Click to see solution

```sql
/* Without INNER JOIN */
SELECT AVG(Price), Manufacturers.Name
  FROM Products, Manufacturers
  WHERE Products.Manufacturer = Manufacturers.Code
  GROUP BY Manufacturers.Name;

/* With INNER JOIN */
SELECT AVG(Price), Manufacturers.Name
  FROM Products INNER JOIN Manufacturers
  ON Products.Manufacturer = Manufacturers.Code
  GROUP BY Manufacturers.Name;
```

14. Select the names of manufacturer whose products have an average price larger than or equal to $150.

Click to see solution

```sql
/* Without INNER JOIN */
SELECT AVG(Price), Manufacturers.Name
  FROM Products, Manufacturers
  WHERE Products.Manufacturer = Manufacturers.Code
  GROUP BY Manufacturers.Name
  HAVING AVG(Price) >= 150;
```

```sql
/* With INNER JOIN */
SELECT AVG(Price), Manufacturers.Name
  FROM Products INNER JOIN Manufacturers
  ON Products.Manufacturer = Manufacturers.Code
  GROUP BY Manufacturers.Name
  HAVING AVG(Price) >= 150;
```

15. Select the name and price of the cheapest product.

Click to see solution

```sql
SELECT name,price
 FROM Products
 ORDER BY price ASC
 LIMIT 1

/* With a nested SELECT */
/* WARNING: If there is more than one item with the cheapest
price it will select them both */
 SELECT Name, Price
   FROM Products
   WHERE Price = (SELECT MIN(Price) FROM Products);
```

16. Select the name of each manufacturer along with the name and price of its most expensive product.

Click to see solution

```sql
/* With a nested SELECT and without INNER JOIN */
  SELECT A.Name, A.Price, F.Name
  FROM Products A, Manufacturers F
  WHERE A.Manufacturer = F.Code
    AND A.Price =
    (
      SELECT MAX(A.Price)
        FROM Products A
        WHERE A.Manufacturer = F.Code
    );

/* With a nested SELECT and an INNER JOIN */
  SELECT A.Name, A.Price, F.Name
  FROM Products A INNER JOIN Manufacturers F
  ON A.Manufacturer = F.Code
    AND A.Price =
    (
      SELECT MAX(A.Price)
        FROM Products A
        WHERE A.Manufacturer = F.Code
    );
```

17. Select the name of each manufacturer which have an average price above $145 and contain at least 2 different products.

Click to see solution

```sql
Select m.Name, Avg(p.price) as p_price, COUNT(p.Manufacturer) as
m_count
FROM Manufacturers m, Products p
WHERE p.Manufacturer = m.code
GROUP BY m.Name , p.Manufacturer
HAVING Avg(p.price) >= 150 and COUNT(p.Manufacturer) >= 2;
```

18. Add a new product: Loudspeakers, $70, manufacturer 2.

Click to see solution

```
INSERT INTO Products( Code, Name , Price , Manufacturer)
 VALUES ( 11, 'Loudspeakers' , 70 , 2 );
```

19. Update the name of product 8 to "Laser Printer".

Click to see solution

```
UPDATE Products
  SET Name = 'Laser Printer'
  WHERE Code = 8;
```

20. Apply a 10% discount to all products.

Click to see solution

```
UPDATE Products
  SET Price = Price - (Price * 0.1);
```

21. Apply a 10% discount to all products with a price larger than or equal to $120.

Click to see solution

```
UPDATE Products
  SET Price = Price - (Price * 0.1)
  WHERE Price >= 120;
```

Table creation code

```
CREATE TABLE Manufacturers (
        Code INTEGER PRIMARY KEY NOT NULL,
        Name CHAR(50) NOT NULL
);

CREATE TABLE Products (
        Code INTEGER PRIMARY KEY NOT NULL,
        Name CHAR(50) NOT NULL ,
        Price REAL NOT NULL ,
        Manufacturer INTEGER NOT NULL
                CONSTRAINT fk_Manufacturers_Code REFERENCES Manufacturers(Code)
);
```

Please note the syntax presented here is for the SQLite system. It has been tested by the authors on `sqlite3` specifically.

The code is also tested against SQL Server 2017.

Also note that the `NOT NULL` constraint on the primary key fields is semantically redundant, but a syntactic necessity in SQLite.

Click to see MySQL syntax.

```
CREATE TABLE Manufacturers (
  Code INTEGER,
  Name VARCHAR(255) NOT NULL,
  PRIMARY KEY (Code)
);

CREATE TABLE Products (
  Code INTEGER,
  Name VARCHAR(255) NOT NULL ,
  Price DECIMAL NOT NULL ,
  Manufacturer INTEGER NOT NULL,
  PRIMARY KEY (Code),
  FOREIGN KEY (Manufacturer) REFERENCES Manufacturers(Code)
) ENGINE=INNODB;
```

Sample dataset

```
INSERT INTO Manufacturers(Code,Name) VALUES (1,'Sony'),(2,'Creative Labs'),(3,'Hewlett-
Packard'),(4,'Iomega'),(5,'Fujitsu'),(6,'Winchester');
```

```
INSERT INTO Products(Code,Name,Price,Manufacturer) VALUES(1,'Hard drive',240,5),
(2,'Memory',120,6),(3,'ZIP drive',150,4),(4,'Floppy disk',5,6),(5,'Monitor',240,1),
(6,'DVD drive',180,2),(7,'CD drive',90,2),(8,'Printer',270,3),(9,'Toner cartridge',66,3),
(10,'DVD burner',180,2);
```