Creating a Relational Database for a fitness retail store

By Jesus Medina

Learning to create a relational database for a non-technical person may seem a very complex task, but after starting to learn a little about SQL and databases, I've realized that the main challenge to creating a database is not to know the code to be able to do it, is to understand the needs of the business and how the data will be used to be able to develop a database that fulfills the business requirements.

In this article, I'll show you how I created my first relational database and the process I used to convert business needs into a simple but effective system to manage daily operations.

For this, we are working with "Fitflex" a retail store, that specializes in selling fitness-related products such as gym equipment, activewear, supplements, and accessories. Why is this important? Because to be able to create a database we need to understand the business.

Here are some facts about Fitflex that are important to consider:

- Fitflex has many store locations (similar to Sportcheck).
- Fitflex sells products in different fitness categories.
- Payments of the orders can be made using cash, credit cards, gift cards, etc.

This project needs to have a mission, and it is **to design a secure, scalable, and efficient database for FitFlex to optimize operations, enhance decision-making, and drive business growth in the fitness retail space**.

When designing a database, it is important that the database and the data stored in it help the business to make data-driven decisions and that the information can be presented and analyzed easily without too much effort.

To start the database design, the first step is to identify the fields and tables. This involves understanding the entities involved in the business process and the attributes that describe them. We'll then organize these into tables, which represent the core data elements, and define the relationships between them.

To start the database design, the first step is to identify the fields and tables. This involves understanding the entities involved in the business process and the attributes that describe them. We'll then organize these into tables, which represent the core data elements, and define the relationships between them.

Here's how I approached this for Fitflex:

1. **Identifying Entities and Attributes:**

I considered the key entities involved in Fitflex's operations. These are the main "things" we need to store information about:

Stores: Fitflex has multiple locations. We need to track information about each store.

Products: Fitflex sells a variety of fitness products. We need details about each product.

Customers: Fitflex interacts with customers who purchase products. We need to store customer information.

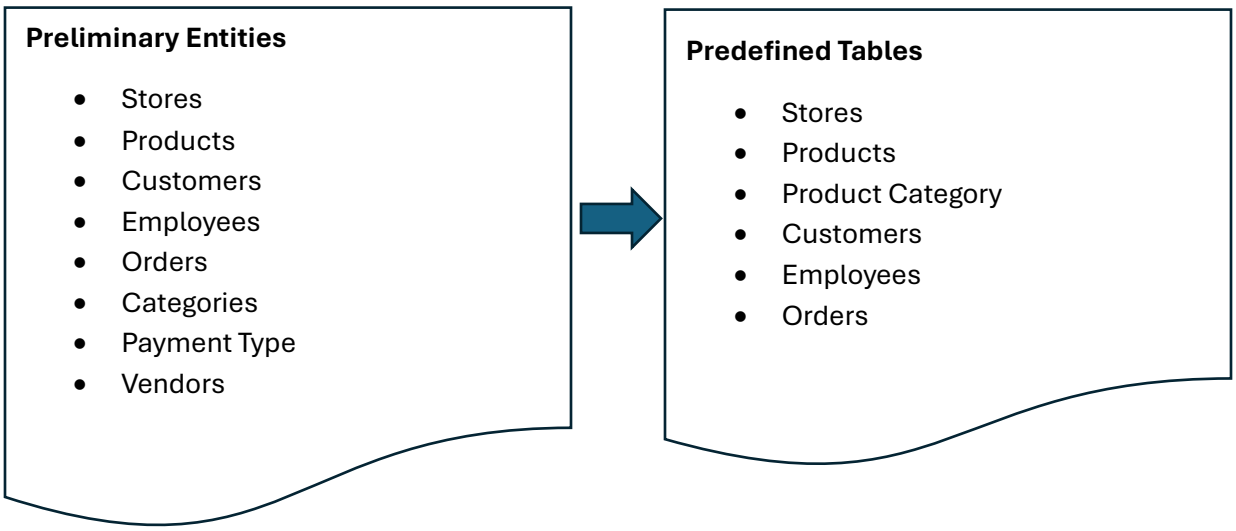
Employees: Fitflex has employees who oversee processing the orders.

Orders: Customers place orders for products. We need to record order details.

Categories: Products are categorized into different fitness categories.

Payments: Orders are paid for using various methods.

There are several other elements that play an important role in the regular operations of Fitflex, such as vendors or suppliers and inventory management, among others, but for the purpose of this article, this will be out of scope. I want to focus on creating a simple but effective database to document the sales transactions.



For each entity, I had to identify the relevant attributes or characteristics.

- Stores
    - StoreName
    - Address
    - Phone
- Products
    - ProductName
    - Price
    - Product Category
- Product Category
    - CategoryName
- Customer
    - FirstName
    - LastName
    - Email
    - Phone
    - Address
- Employee
    - FirstName
    - LastName
    - Email
    - Phone
    - Address
    - Role
- Order
    - Order Number
    - Product
    - ProductQty
    - Employee
    - Customer
    - Store
    - Date
    - Status
    - Payment Type

2. Creating Tables

Now that we have identified the entities and attributes, the tables can be created. For each table, a Primary key must be identified. This will ensure that each record on the table is unique and that we don't have duplicates. Also, in this step, we can add validation rules for each attribute on the table. These validation rules are going to indicate the type of data that will be stored.

Store Table

Field	Data type
StoreID (PK)	Integer
StoreName	Varchar
Address	Varchar
Phone	Varchar

Product Table

Field	Data type
ProductID (PK)	Integer
ProductName	Varchar
Price	Decimal
CategoryID	Varchar

Product Category Table

Field	Data type
CategoryID (PK)	Integer
CategoryName	Varchar

Customer Table

Field	Data type
CustomerID (PK)	Integer
FirstName	Varchar
LastName	Varchar
Email	Varchar
Address	Varchar
Phone	Varchar

Employee Table

Field	Data type
EmployeeID (PK)	Integer
FirstName	Varchar
LastName	Varchar
Email	Varchar
Address	Varchar
Phone	Varchar
Role	Varchar

Payment Mode Table

Field	Data type
PaymentID (PK)	Integer
PaymentType	Varchar

Sales Order Table

Field	Data type
OrderID (PK)	Integer
ProductID (PK)	Integer
ProductQty	Integer
CustomerID	Integer
EmployeeID	Integer
StoreID	Integer
OrderDate	Date
StatusID	Integer
PaymentID	Integer

In almost every table there is a unique Primary Key, but in this case, for the sales order table, we are using a composite primary key, using as fields for this composite key, the OrderID and the ProductID. This combination ensures there are no duplicates in the rows and we can handle the relationships appropriately.

3. **Defining Relationships**

The next step is to define the relationship between the tables. I have defined the following relationships:

One-to-many relationships:

- Customers → SalesOrders (One customer can place multiple orders)

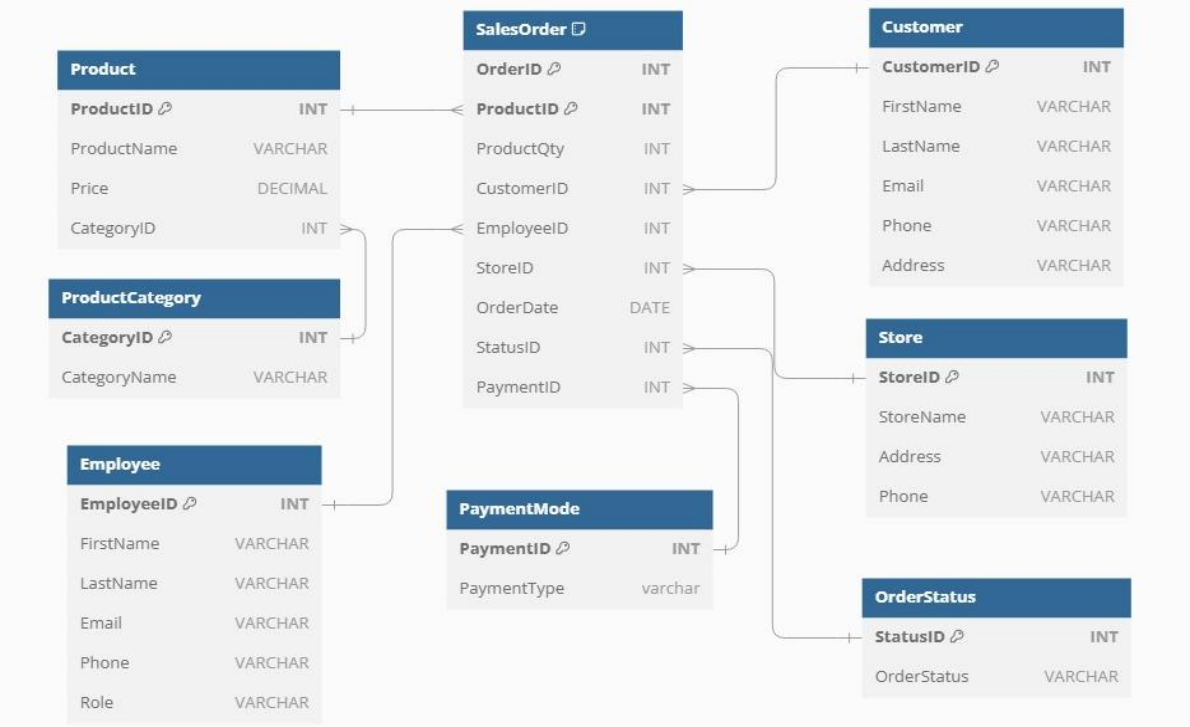
- SalesOrders → Products (One order can have multiple products)
- SalesOrders → Store (One order is placed at a single store, but a store can process multiple orders)
- ProductCategory → Products (One category can have multiple products)
- PaymentMode → SalesOrders (One payment method can be used in multiple orders)
- Store → SalesOrders (One store can have multiple orders)
- OrderStatus → SalesOrders (One order status can be assigned to multiple orders)

Many-to-Many Relationship & Solution:

- Products & SalesOrders have a many-to-many relationship (one order can contain multiple products, and a product can appear in multiple orders).

To handle this, we created the SalesOrder table. This table will have a composite primary key with OrderID and ProductID being part of this composite key. This will work in a way similar to a linking table and ensure there are no duplicate entries in the table. In case there is more than one of the same product, this is handled in the ProductQty field of the table. There is no occasion where we will have a repeated entry with the same OrderID and ProductID combination.

With all these tables and relationships now defined, we can build our Entity Relationship diagram showing how all the tables are interconnected.



Now, we can create our database in Azure Data Studio. One important thing to take into consideration when creating the database is that we must declare not only the Primary Keys on the table but also the Foreign Keys. This is how the database software determines the relationship between the tables. This is the code we used to create the Fitflex database.

```
-- Creating and selecting the Database
CREATE DATABASE Fitflex
USE Fitflex
-- Creating the OrderStatus table
CREATE TABLE OrderStatus (
    StatusID INT PRIMARY KEY AUTO_INCREMENT,
    OrderStatus VARCHAR(255) NOT NULL
);
-- Creating the Employee table
CREATE TABLE Employee (
    EmployeeID INT PRIMARY KEY AUTO_INCREMENT,
    FirstName VARCHAR(255) NOT NULL,
    LastName VARCHAR(255) NOT NULL,
    Email VARCHAR(255) NOT NULL,
    Phone VARCHAR(15),
    Role VARCHAR(50)
);
-- Creating the Customer table
CREATE TABLE Customer (
    CustomerID INT PRIMARY KEY AUTO_INCREMENT,
    FirstName VARCHAR(255) NOT NULL,
    LastName VARCHAR(255) NOT NULL,
    Email VARCHAR(255) NOT NULL,
    Phone VARCHAR(15),
    Address VARCHAR(255)
);
-- Creating the ProductCategory table
```

```
CREATE TABLE ProductCategory (
    CategoryID INT PRIMARY KEY AUTO_INCREMENT,
    CategoryName VARCHAR(255) NOT NULL
);
-- Creating the Product table
CREATE TABLE Product (
    ProductID INT PRIMARY KEY AUTO_INCREMENT,
    ProductName VARCHAR(255) NOT NULL,
    Price DECIMAL(18,2) NOT NULL,
    CategoryID INT,
    FOREIGN KEY (CategoryID) REFERENCES ProductCategory(CategoryID)
);
-- Creating the Store table
CREATE TABLE Store (
    StoreID INT PRIMARY KEY AUTO_INCREMENT,
    StoreName VARCHAR(255) NOT NULL,
    Address VARCHAR(255) NOT NULL,
    Phone VARCHAR(15)
);
-- Creating the PaymentMode table
CREATE TABLE PaymentMode (
    PaymentID INT PRIMARY KEY AUTO_INCREMENT,
    PaymentType VARCHAR(50) NOT NULL
);
-- Creating the SalesOrder table
CREATE TABLE SalesOrder (
    OrderID INT,
    ProductID INT,
    ProductQty INT NOT NULL,
    CustomerID INT,
    EmployeeID INT,
```

```
    StoreID INT,
    OrderDate DATE NOT NULL,
    StatusID INT,
    PaymentID INT,
    PRIMARY KEY (OrderID, ProductID),
    FOREIGN KEY (ProductID) REFERENCES Product(ProductID),
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID),
    FOREIGN KEY (EmployeeID) REFERENCES Employee(EmployeeID),
    FOREIGN KEY (StoreID) REFERENCES Store(StoreID),
    FOREIGN KEY (StatusID) REFERENCES OrderStatus(StatusID),
    FOREIGN KEY (PaymentID) REFERENCES PaymentMode(PaymentID)
);
```

Now that the database is available, we can start using it to store business data in an organized way. It is at this precise point that the value of a database is unmeasurable because we can use it to have business insights and make data-driven decisions.

To help in this process of making data-driven decisions we can create views. Views are just tables created to help visualize certain key aspects of the business. Some views that can be created that provides a lot of value are:

- Order details (similar to receipt)
- Top 5 best selling products.
- Top 5 best selling employees.
- Order status in a specific store.
- Best performing store.

These are just a few ideas for views we can have from our database. For this case study I have created 2 views.

The benefit of the view is that it's code is stored in the database and with a simple line of code we can have displayed the view information. This helps save a lot of time and effort.

a. Order details (Customer receipt)

Purpose: want to show the customer the details of their order

Tables used: SalesOrder, Customer, Employee, Product, Store, OrderStatus, PaymentMode

View Name: OrderDetails

Fields Used: CustomerName, EmployeeName, StoreName, OrderID, OrderStatus, PaymentType, ProductName, ProductQty, OrderTotal

Calculated Fields: ProductTotal, OrderTotal, Customer Name, EmployeeName

```
352  CREATE VIEW OrderDetails AS
353  SELECT
354      so.OrderID,
355      CONCAT(c.FirstName, ' ', c.LastName) AS CustomerName,
356      CONCAT(e.FirstName, ' ', e.LastName) AS EmployeeName,
357      s.StoreName,
358      os.OrderStatus,
359      pm.PaymentType,
360      p.ProductName,
361      so.ProductQty,
362      (p.Price * so.ProductQty) AS ProductTotal,
363      (SELECT SUM(p2.Price * so2.ProductQty)
364       FROM SalesOrder so2
365       JOIN Product p2 ON so2.ProductID = p2.ProductID
366       WHERE so2.OrderID = so.OrderID) AS OrderTotal
367  FROM
368      SalesOrder so
369  JOIN
370      Customer c ON so.CustomerID = c.CustomerID
371  JOIN
372      Employee e ON so.EmployeeID = e.EmployeeID
373  JOIN
374      Store s ON so.StoreID = s.StoreID
375  JOIN
376      OrderStatus os ON so.StatusID = os.StatusID
377  JOIN
378      PaymentMode pm ON so.PaymentID = pm.PaymentID
379  JOIN
380      Product p ON so.ProductID = p.ProductID;
381
382  SELECT * FROM OrderDetails WHERE OrderID = 2;
```

Results Messages

	OrderID	CustomerName	EmployeeName	StoreName	OrderStatus	PaymentType	ProductName	ProductQty	ProductTotal	OrderTotal
1	2	Dwight Schrute	Jane Smith	Uptown Fitness	Completed	Credit Card	Resistance Bands	3	59.97	2609.76
2	2	Dwight Schrute	Jane Smith	Uptown Fitness	Completed	Credit Card	Running Shoes	7	629.93	2609.76
3	2	Dwight Schrute	Jane Smith	Uptown Fitness	Completed	Credit Card	Hiking Backpack	9	1169.91	2609.76
4	2	Dwight Schrute	Jane Smith	Uptown Fitness	Completed	Credit Card	Fitness Tracker	5	749.95	2609.76

b. Top 5 Best Selling Products

Purpose: Identify the top 5 selling products

Tables used: SalesOrder, Product

View Name: Hot Selling Products

Fields Used: ProductID, ProductName

Calculated Fields: TotalQuantitySold, TotalSales

```
384   CREATE VIEW HotProducts AS
385   SELECT
386       p.ProductID,
387       p.ProductName,
388       SUM(so.ProductQty) AS TotalQuantitySold,
389       SUM(so.ProductQty * p.Price) AS TotalSales
390   FROM
391       SalesOrder so
392   JOIN
393       Product p ON so.ProductID = p.ProductID
394   GROUP BY
395       p.ProductID, p.ProductName
396   ORDER BY
397       TotalSales DESC
398   LIMIT 5;
399
400   SELECT * FROM HotProducts
```

Results Messages

	ProductID	ProductName	TotalQuantitySold	TotalSales
1	1	Treadmill	28	27999.72
2	10	Elliptical Machine	15	8999.85
3	7	Hiking Backpack	28	3639.72
4	8	Fitness Tracker	21	3149.79
5	2	Dumbbell Set	15	2999.85

Conclusion

This database represents a significant step forward for Fitflex, providing a robust foundation for managing daily operations. It allows us to efficiently record and track sales transactions, manage customer information, and generate reports. Beyond simple record-keeping, this system facilitates data-driven decision-making by centralizing and organizing key business data. This allows us to monitor performance metrics, identify emerging trends, and gain a deeper understanding of customer behavior. The ability to easily extract and present data in clear, concise reports is of a great value for the organization.

While this initial version focuses on the sales transaction process, laying a solid groundwork for managing customer orders, payments, and product details, it's designed for future growth. This database can evolve into a comprehensive system that encompasses all aspects of Fitflex's operations. Future development will prioritize integrating crucial functionalities such as inventory management and vendor relationships.

The relational structure of this database is key to its scalability and adaptability. This design allows integration of new features and data as the business expands and its needs change. This ensures that future additions can be incorporated without requiring a complete redesign, protecting our initial investment and providing a flexible platform for continued growth and improvement.