E-SHOP DATABASE DESIGN AND IMPLEMENTATION

DATA DESIGN MANAGEMENT

OLUBUNMI ELIZABETH ALBERT-ABU

DATABASE DESIGN AND IMPLEMENTATION WITH MySQL

24TH JANUARY, 2024

DECLARATION

I certify that this report is entirely my original work and I have fully referenced and correctly cited the work of others, where required.

TABLE OF CONTENTS

EXECUTIVE SUMMARY

This presentation sets out to describe the process of designing a robust database for an E-shop whose operation is grounded in the global sales of music records, movies and books. The schema design was structured to capture all entities and their attributes as well as support efficient data retrieval and management. The Entity Relationship Diagram defines clear relationships and cardinalities between the different tables while, SQL queries demonstrated effective retrieval of key information. Designing a well-structured schema lays the foundation for effective database management and can pose some complexities at the development, normalization and query formulation phase. Such challenges were tackled and thus, aided a further integration of a scalable system for anticipated growth and expansion.

INTRODUCTION

Database design plays a pivotal role in ensuring seamless operations, efficient data organization and optimal performance in digital commerce platforms (Knaflic and Cole Nussbaumer, 2015), like the E-shop described in this use case. An organized and efficient database facilitates streamlined customer experiences and effective business management.

It builds up from well-defined entities, their attributes, existing relationships and connected cardinalities which is showcased by an Entity Relationship Diagram.

The E-shop, herein named **'Virtualshop'**, functions as a global marketplace for music records, movies and books. It encompasses vital components like customer data, employee management, diverse products, order processing and payment transactions.

To perfectly address the tasks at hand, an in-depth understanding of Database Management, the Use Case and Entity Relationship Diagram will be further discussed.

USE CASE OVERVIEW

The use case envisions an interactive user-friendly interface that facilitates a straightforward experience for customers, streamlining their search for entertainment media and ensuring hassle-free transactions from selection to delivery. The E-shop covers the following functionalities:

- Key components of the E-shop are customers, employees, products, orders and payments.
- Customer interacts with the digital platform to browse, select and purchase music records, movies and books. Orders are paid for by several means including credit cards.
- Employees oversee product management, order processing, deliveries, accounts and human resources. An employee can only work in one department of the business.
- The departments existing are Deliveries, Accounting and Human Resources. Each department can have multiple employees.
- The E-shop has a global reach catering to customers via orders and deliveries to different countries.

ENTITY RELATIONSHIP (ER) DIAGRAM

An Entity-Relationship Diagram is a visual representation of relationships between entities in a database (Abraham et al, 2019). This graphical overview aids the designing and implementation of a well-structured database system. It helps database designers, developers and stakeholders understand the database structure, its entities, attributes and how they relate to each other.

- Entities are distinct objects in a database such as customers, products and orders.
- Attributes are properties of entities that describes specific details such as a customer's name or age.
- Relationships depicts how entities are interconnected such as one that exists between customers and orders. Straight and dotted lines are used to signify strong and weak relationships respectively.
- Cardinality defines the numerical relationship between entities in a relationship. It is represented by symbols and statements like 'one-to-one', 'one-to-many' and 'many-to-many'.

IMPORTANCE OF ER DIAGRAM FOR DATABASE DESIGN

- It aids visualization and understanding of the database structure.
- It depicts relationships clearly.
- it serves as a basis for database schema and tables creation.
- It ensures data integrity by aiding identification of areas that requires optimization and normalization.
- It facilitates effective communication among the database stakeholders, developers and designers.

ENTITY RELATIONSHIP (ER) DIAGRAM

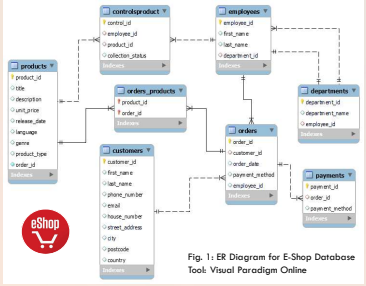


Fig. 1: ER Diagram for E-Shop Database
Tool: Visual Paradigm Online

The ER Diagram in Fig. 1 shows the interconnectedness between the entities within the E-shop schema.

The following relationships exists as seen:

ONE-TO-ONE

- An employee can only belong to one department.
- A customer can make one order.

ONE-TO-MANY

- Each customer can place multiple orders.
- A customer can use different payment methods.
- A department can have multiple employees.
- An order can contain various products.

MANY-TO-ONE

- Multiple employees can exist in a department.
- Several orders can belong to one customer.
- Various products can be in one order.
- Many products can be processed by one employee.

MANY-TO-MANY

- Multiple employees can control products for delivery and various products can be processed by several employees.
- Orders can include multiple products and products can be part of various orders.

DATABASE MANAGEMENT

Database management involves the systematic organization, storage, retrieval and manipulation of data within a database system.

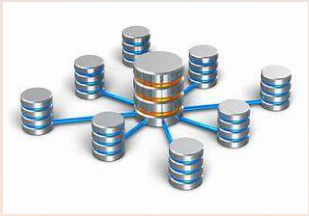


Fig. 2: Sample Database Structure

DATABASE MANAGEMENT - NOTES

The database structure as depicted in Fig. 1, can be updated and managed to ensure data security, integrity and efficient access.
There are Database Management Systems (DBMS) that provides a software environment that manages databases. According to Stackshare (2024), MySQL is one of the best Relational Database Management Systems that stores and manages structured data, and is the tool used in the development and visualization of this E-shops' schema.

In the context of the E-shop use case, creating a database or schema involves defining the structure of tables to store information about the entities which are customers, products, employees, departments, orders and payments. In addition to these entities from the use case, are new ones that depicts the many-to-many relationships existent in the database and they are created with 'junction tables' which joins two 'one-to-many' relationships in a middle table (Nicola A. Amato, 2023).

Some other vital components of entities that aids the database schema creation and populations are:

- *Primary keys which uniquely identifies important records in a table e.g. customer_id, product-id.*
- *Foreign keys which establish relationships between tables e.g. customer_id in Orders table.*
- *Constraints defines rules for data values in a table e.g. NOT NULL, UNIQUE.*

E-SHOP SCHEMA CREATION

Below is a detailed process of the E-shop schema creation and population implemented by Structured Query Language (SQL) scripts using the MySQL workbench 8.2 software. Careful attention was given to the selection of the sample data as specific criteria had been given in the use case.

SQL SCRIPTS

First step in this process is to create the database which I named 'Virtualshop'. Fig. 3 below depicts the script that does this.

Fig. 3:

```
CREATE DATABASE virtualshop;
```

Next, the Customers table, its attributes, primary key and data types was created as shown in Fig. 4.

Fig. 4:

E-SHOP SCHEMA CREATION – SQL SCRIPTS

Fig. 5 below depicts the population of the Customers table with sample data.

Fig. 5:

```
INSERT INTO Customers (customer_id, first_name, last_name, phone_number, email, house_number, street_address, city, postcode, country)
VALUES
    ('C0001', 'Joy',    'Bob',  '07065464324',  'joy@shopme.com',   '22b',  'Bridge Avenue',    'Kent', '0921', 'UK'),
    ('C0002', 'Samuel', 'Mason',    '07675467368',  'samuel@shopme.com',    '87a',  'Apex Street',  'Bristol',  '1290', 'UK'),
    ('C0003', 'Jane',   'George',   '07568584373',  'jane@shopme.com',  '91',   'Gate Road',    'Kent', '0831', 'UK'),
    ('C0004', 'Joshua', 'Ben',  '07689706473',  'joshua@shopme.com',    '897',  'Glory Lane',   'Manchester',   '4576', 'USA'),
    ('C0005', 'Samantha',   'Grace',    '07953626578',  'samantha@shopme.com',  '56',   'Peace Street', 'Bristol',  '1378', 'UK');
```

Using `SELECT * FROM virtualshop.customers;` *the Customers table can be visualized as seen in fig. 6 below.*

Fig. 6:

customer_id	first_name	last_name	phone_number	email	house_number	street_address	city	postcode	country
C0001	Joy	Bob	07065464324	joy@shopme.com	22b	Bridge Avenue	Kent	0921	UK
C0002	Samuel	Mason	07675467368	samuel@shopme.com	87a	Apex Street	Bristol	1290	UK
C0003	Jane	George	07568584373	jane@shopme.com	91	Gate Road	Kent	0831	UK
C0004	Joshua	Ben	07689706473	joshua@shopme.com	897	Glory Lane	Manchester	4576	UK
C0005	Samantha	Grace	07953626578	samantha@shopme.com	56	Peace Street	Bristol	1378	UK

As seen in fig. 6, the Customers table was generated and is complete.

E-SHOP SCHEMA CREATION – SQL SCRIPTS

Fig. 7:

```sql
CREATE TABLE Products (
    product_id VARCHAR(50) PRIMARY KEY,
    title VARCHAR(100),
    description TEXT,
    unit_price DECIMAL(10, 2),
    release_date DATE,
    language VARCHAR(50),
    genre VARCHAR(50),
    product_type VARCHAR(50));
```

Fig. 8:

```sql
INSERT INTO Products (product_id, title, description, unit_price, release_date, language, genre, product_type)
VALUES
    ('P0001', 'I Trend!', 'Classic Sci-Fi movie', 18.99, '2001-09-01', 'English', 'Science Fiction', 'Movie'),
    ('P0002', 'The Scroll', 'Dystopian novel by George Orwell', 9.99, '1999-05-11', 'English', 'Dystopian', 'Book'),
    ('P0003', 'Trust', 'Ben Graham album', 19.99, '2015-07-07', 'English', 'Pop', 'Music'),
    ('P0004', 'Inception', 'Mind-bending thriller', 14.99, '2016-10-07', 'English', 'Thriller', 'Movie'),
    ('P0005', 'The Hobbit', 'Fantasy novel by J.R.R. Tolkien', 24.99, '2011-09-20', 'English', 'Fantasy', 'Book');
```

Using `SELECT * FROM virtualshop.products;` *the Products table can be visualized as seen in fig. 6 below.*

Fig. 9:

product_id	title	description	unit_price	release_date	language	genre	product_type
P0001	I T...	Classic Sc...	18.99	2001-09-01	English	Science Fiction	Movie
P0002	Th...	Dystopia...	9.99	1999-05-11	English	Dystopian	Book
P0003	Trust	Ben Grah...	19.99	2015-07-07	English	Pop	Music
P0004	In...	Mind-ben...	14.99	2016-10-07	English	Thriller	Movie
P0005	Th...	Fantasy ...	24.99	2011-09-20	English	Fantasy	Book

E-SHOP SCHEMA CREATION – NOTES

Adhering to the Use case specifications, the same process is repeated for the Products table as seen in Fig. 7 – 9 above. Taking into consideration the strong many-to-many relationship that exists between products and orders, the need for the creation of a junction table that normalizes this relationship between them arose and can be seen in the ER Diagram as shown in fig. 1.

E-SHOP SCHEMA CREATION – SQL SCRIPTS

The same process is repeated for the Orders table as seen in Fig. 10 – 12 below.

Fig. 10:

```
CREATE TABLE Orders (
    order_id VARCHAR(50) PRIMARY KEY,
    customer_id VARCHAR(50),
    product_id VARCHAR(50),
    order_date DATE,
    payment_method VARCHAR(50),
    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id),
    FOREIGN KEY (product_id) REFERENCES Products(product_id));
```

Fig. 11:

```
INSERT INTO Orders (order_id, customer_id, product_id, order_date, payment_method)
VALUES
    ('OR0001', 'C0001', 'P0001', '2022-11-05', 'Credit Card'),
    ('OR0002', 'C0002', 'P0002', '2022-10-20', 'Paypal'),
    ('OR0003', 'C0003', 'P0003', '2022-10-24', 'Credit Card'),
    ('OR0004', 'C0001', 'P0004', '2022-10-28', 'Gift Card'),
    ('OR0005', 'C0002', 'P0005', '2022-11-02', 'Loyalty Card');
```

Using `SELECT * FROM virtualshop.orders;` *the Orders table can be visualized as seen in fig. 9 below.*

Fig. 12:

order_id	customer_id	product_id	order_date	payment_method	employee_id
OR0001	C0001	P0001	2022-11-05	Credit Card	EM001
OR0002	C0002	P0002	2022-10-20	Paypal	EM005
OR0003	C0003	P0003	2022-10-24	Credit Card	EM001
OR0004	C0001	P0004	2022-10-28	Gift Card	EM005
OR0005	C0002	P0005	2022-11-02	Loyalty Card	EM001

E-SHOP SCHEMA CREATION – NOTES

The orders entity bears relationships with customers, products and employees, hence the inclusion of those entity's primary keys in its table.

In this table, they are known as foreign keys and this places a constraint that aids seamless retrieval of information from the database. In particular, the product_id key in this table fulfills the 2 use case scenarios that states ' A customer can place orders for products' and ' A customer can make multiple orders for products'. This creates a normalized junction table as explained previously that defines the many-to-many relationship between products and orders as seen in the ER Diagram in fig. 1.

E-SHOP SCHEMA CREATION – SQL SCRIPTS

DEPARTMENTS

Fig. 13:

```
CREATE TABLE Departments
    (department_id VARCHAR(50) PRIMARY KEY,
    department_name VARCHAR(50));
```

Fig. 14:

```
INSERT INTO Departments (department_id, department_name)
VALUES
    ('DELVS', 'Deliveries'),
    ('ACCT', 'Accounting'),
    ('HR', 'Human Resources');
```

Fig. 15:

```
ALTER TABLE Departments
ADD COLUMN employee_id VARCHAR(50);
ALTER TABLE Departments
ADD FOREIGN KEY (employee_id) REFERENCES Employees(employee_id);
```

```
SELECT * FROM virtualshop.departments;
```

Fig. 16:

department_id	department_name	employee_id
ACCT	Accounting	EM002
DELVS	Deliveries	EM001, EM005
HR	Human Resources	EM003, EM004

EMPLOYEES

Fig. 17

```
CREATE TABLE Employees
    (employee_id VARCHAR(50) PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    department_id VARCHAR(50),
    FOREIGN KEY (department_id) REFERENCES Departments(department_id));
```

Fig. 18

```
INSERT INTO Employees (employee_id, first_name, last_name, department_id)
VALUES
    ('EM001', 'Micah', 'Homes', 'DELVS'),
    ('EM002', 'Sarah', 'Williams', 'ACCT'),
    ('EM003', 'Chris', 'Buyer', 'HR'),
    ('EM004', 'Martha', 'Simpson', 'HR'),
    ('EM005', 'Fred', 'Davidson', 'DELVS');
```

```
SELECT * FROM virtualshop.employees;
```

Fig. 19

employee_id	first_name	last_name	department_id
EM001	Micah	Homes	DELVS
EM002	Sarah	Williams	ACCT
EM003	Chris	Buyer	HR
EM004	Martha	Simpson	HR
EM005	Fred	Davidson	DELVS

E-SHOP SCHEMA CREATION – NOTES

The same process applies to the Departments and Employees tables, fig. 16 and 19 above, but with careful consideration of the use case scenario that explicitly states 'Each employee is registered to only one department, while a department can have multiple employees'.

This points to a one-to-one and one-to-many relationship, hence, It was imperative to alter my departments table to include the employee identifier as a foreign key as seen in fig. 15.

The use case also specified that at least one employee's first name should start with capital letter S, and this also guided the naming of my sample data.

E-SHOP SCHEMA CREATION – SQL SCRIPTS

PAYMENTS

```
CREATE TABLE Payments
    (payment_id INT PRIMARY KEY AUTO_INCREMENT,
    order_id VARCHAR(50),
    payment_method VARCHAR(50),
    FOREIGN KEY (order_id) REFERENCES Orders (order_id));
```

Fig. 20:

```
INSERT INTO Payments (payment_id, order_id, payment_method)
VALUES
    ('1', 'OR0001', 'Credit Card'),
    ('2', 'OR0002', 'Paypal'),
    ('3', 'OR0003', 'Credit Card'),
    ('4', 'OR0004', 'Cash'),
    ('5', 'OR0005', 'Loyalty Card');
```

Fig. 21:

```
SELECT * FROM virtualshop.payments;
```

payment_id	order_id	payment_method
1	OR0001	Credit Card
2	OR0002	Paypal
3	OR0003	Credit Card
4	OR0004	Cash
5	OR0005	Loyalty Card

Fig. 22:

CONTROLSPRODUCT (JUNCTION TABLE)

```
CREATE TABLE ControlsProduct
    (control_id CHAR(50) PRIMARY KEY,
    employee_id VARCHAR(50),
    product_id VARCHAR(50),
    collection_status VARCHAR(50),
    FOREIGN KEY (employee_id) REFERENCES Employees (employee_id),
    FOREIGN KEY (product_id) REFERENCES Products (product_id));
```

Fig. 23

```
INSERT INTO ControlsProduct (control_id, employee_id, product_id, collection_status)
VALUES
    ('CP1', 'EM001', 'P0002', 'Collected'),
    ('CP2', 'EM001', 'P0004', 'Pending'),
    ('CP3', 'EM001', 'P0003', 'Dispatched'),
    ('CP4', 'EM001', 'P0001', 'Delivered'),
    ('CP5', 'EM001', 'P0005', 'Pending');
```

Fig. 24

```
SELECT * FROM virtualshop.controlsproduct;
```

control_id	employee_id	product_id	collection_status
CP1	EM001	P0002	Collected
CP2	EM001	P0004	Pending
CP3	EM001	P0003	Dispatched
CP4	EM001	P0001	Delivered
CP5	EM001	P0005	Pending

Fig. 25

E-SHOP SCHEMA CREATION – NOTES

The Payments table as seen in fig. 22, is necessary as it helps to fulfill the scenario where a customer can make payments using credit card. This means the E-shop also uses other payment methods which some samples have been captured in its table. The 'order_id' is a foreign key in this table as there is a one-to-many relationship between payments and orders.

The ControlsProduct table shown in fig. 25 above, is called a junction table because it normalizes the many-to-many relationship between employees and products by joining their respective one-to-many relationships together in a new table by utilizing their primary keys as seen in the ER Diagram shown in fig. 1. The collection_status attribute in this table depicts the stage a product or an order is at in its delivery process, from collection to delivered to the customer.

SQL QUERIES

As opined by Ignasio L. Bisso (2022), SQL query is a statement that determines the type of data to be retrieved from the created database. These queries can help test the functionality or completeness of the database as well as help the user access information quickly.

Below are some SQL queries and results from the 'Virtualshop' database:

a. Extract all the customers from a specific city:

This SQL Query will retrieve all customers living in a defined city which is **'Kent'** in this instance. This information can aid surveys, analysis, targeted marketing or could be used for any other aim the user requires.

Statement:

```
SELECT * FROM Customers WHERE city = 'Kent';
```

Result:

customer_id	first_name	last_name	phone_number	email	house_number	street_address	city	postcode	country
C0001	Joy	Bob	07065464324	joy@shopme.com	22b	Bridge Avenue	Kent	0921	UK
C0003	Jane	George	07568584373	jane@shopme.com	91	Gate Road	Kent	0831	UK

Fig. 26

SQL QUERIES - NOTES

As seen in fig. 26, there are only two customers in the database that lives in the city 'Kent'.

Other information about these customers can be easily seen.

SQL QUERIES

b. Search for a product of a specific genre:

This SQL Query will retrieve all products of a particular genre as defined in the statement given. In this instance, that genre is **'Fantasy'.**

This information can draw insights into the quantity of products in stock that falls into the fantasy **genre** category. The results can also aid other requirements of the user.

Statement:

```
SELECT * FROM Products WHERE genre = 'Fantasy';
```

Result:

product_id	title	description	unit_price	release_date	language	genre	product_type
P0005	The Hobbit	Fantasy novel by J.R.R. Tolkien	24.99	2011-09-20	English	Fantasy	Book

Fig. 27

SQL QUERIES - NOTES

As seen in fig. 27, there is just one product in the database that falls under the genre category called 'Fantasy'.

Other product information such as title, price and type can also be seen.

SQL QUERIES

c. Count how many customers are from a particular city:

The total number of customers in a chosen city will be retrieved using this SQL Query. In this instance, that city is **'Bristol'**.

The difference between this query and that commanded by the **'Extract'** statement is that this will state the total **number** of customers in that city by a **count** function. This gives a straightforward insight to the number of the specified data.

Statement:

```
SELECT COUNT(*) FROM Customers WHERE city = 'Bristol';
```

Result:

	COUNT(*)
▶	2

Fig. 28

SQL QUERIES - NOTES

As seen in fig. 28, there are 2 customers that are from Bristol in the 'Virtualshop' database using the 'SELECT COUNT' function.

The customers' other information cannot be seen unlike when a similar query was made to extract their details with simply the 'SELECT' function.

SQL QUERIES

d. Calculate the average of the unit price:

The **average** of the total unit price of all **products** in the '**Virtualshop**' database can be calculated using the SQL query **AVG** function. AVG stands for Average. Since the Products table has a column titled **'unit_price'**, this query will be effected without error as long as there are no limiting constraints.

The average price can support any financial or sales projections and analysis the user needs.

Statement:

```
SELECT AVG(unit_price) FROM Products;
```

Result:

	AVG(unit_price)
▶	17.790000

The average unit price of all products in the Product's table within the database is approximately 17.8 as seen in fig. 29.

Fig. 29

SQL QUERIES

e. Extract all current orders:

This command will retrieve all the orders that are currently in the database. Orders are for products that payments have been made for.

Retrieving this information can be for many purposes, one of such may be to aid inventory management.

Statement:

```
SELECT * FROM Orders;
```

Result:

order_id	customer_id	product_id	order_date	payment_method
OR0001	C0001	P0001	2022-11-05	Credit Card
OR0002	C0002	P0002	2022-10-20	Paypal
OR0003	C0003	P0003	2022-10-24	Credit Card
OR0004	C0001	P0004	2022-10-28	Cash
OR0005	C0002	P0005	2022-11-02	Loyalty Card

Fig. 30

SQL QUERIES - NOTES

Fig. 30 shows there are currently 5 orders in the database. The products to which these orders are for can be tracked by use of the product_id and the customer can be known via the customer_id.

Other details such as the payment method and order date are also available.

SQL QUERIES

f. Extract all orders for books that has the keyword 'the' in their description:

This is a unique query to determine an **order** which has a particular keyword, **'the'**, in the products' description.

To achieve this, the function **'JOIN'** will be used to connect the Products and Orders table, utilizing the **'product_id'**, **'product_type'** (**'book'** in this instance) and **'description'** columns, to retrieve the **order** for **books** that has the keyword **'the'**.

Statement:

```
SELECT * FROM Orders
JOIN Products ON Orders.product_id = Products.product_id
WHERE Products.product_type = 'Book' AND Products.description LIKE '%the%';
```

Result:

order_id	customer_id	product_id	order_date	payment_method	product_id	title	description	unit_price	release_date	language	genre	product_type
OR0002	C0002	P0002	2022-10-20	Paypal	P0002	The Scroll	The Dystopian novel by George Orwell	9.99	1999-05-11	English	Dystopian	Book

Fig. 31

SQL QUERIES - NOTES

SQL is enabled to pull out different types of information from a database.

A need for such retrieval can be seen in a situation where a specific order needs to be ascertained or tracked for other details such as the day it was ordered or to have a clear view of all details about a particular order as seen in fig. 31.

SQL QUERIES

g. Extract all payments with credit card for music records:

This is another unique query that will extract **orders** for **music record** that was completed via **credit card payment.**

Statement:

```
SELECT * FROM Payments
JOIN Orders ON Payments.order_id = Orders.order_id
JOIN Products ON Orders.product_id = Products.product_id
WHERE Payments.payment_method = 'Credit Card' AND Products.product_type = 'Music';
```

Result:

payment_id	order_id	payment_method	order_id	customer_id	product_id	order_date	payment_method	product_id	title	description	unit_price	release_date	language	genre	product_type
3	OR0003	Credit Card	OR0003	C0003	P0003	2022-10-24	Credit Card	P0003	Trust	Ben Graham album	19.99	2015-07-07	English	Pop	Music

Fig. 32

SQL QUERIES - NOTES

As seen in fig. 32, the **'JOIN'** function is once again used but now connecting 3 tables, namely Products, Orders and Payments, together to be able to effectively retrieve the required information. The needed columns in these tables to draw out all payments with credit card for music record are order_id, product_id, product_type (music) and payment_method (credit card).

A sample scenario where this query may be useful is when the Accounting department is processing bank reconciliation to balance the E-shop's financial records.

SQL QUERIES

h. Count how many employees handle music records:

Similarly, the **'JOIN'** function is used to combine Orders, Products and Employees tables within the **Virtualshop** database to retrieve the **number** of **employees** that handles **music records.** The columns needed to fulfill this query are **employee_id, product_id and product_type (music).**

Statement:

```
SELECT COUNT(*) FROM Employees
JOIN Orders ON Employees.employee_id = Orders.employee_id
JOIN Products ON Orders.product_id = Products.product_id
WHERE Products.product_type = 'Music';
```

Result:

	COUNT(*)
▶	1

Fig. 33

This query also depicts the versatility of the 'SELECT COUNT' function showing that it can count from combined tables within a schema.

As seen in fig. 33, there is only 1 employee that handles music records.

SQL QUERIES

i. Count the employees with the first name starting with the letter S (S capital):

Here, the number of employees with their first name starting with capital letter S is required.

The **'SELECT COUNT'** function is used to retrieve this information with the SQL query statement below, specifying **'FROM'** the **Employees** table and **'WHERE'** capital letter **'LIKE' S** exists in the **first_name** column.

Statement:

```
SELECT COUNT(*) FROM Employees WHERE first_name LIKE 'S%';
```

Result:

	COUNT(*)
▶	1

Fig. 34

SQL QUERIES - NOTES

There is only 1 employee whose first name starts with capital letter 'S' in the E-shop's database as seen in fig. 34.

A query like this can be used in different scenarios such as when the information is required to identify the employee that attended to a particular customer that noted this attendee's first name started with a capital letter 'S' but cannot recall the full name. This can easily be matched up with the product such a customer ordered, or any other information, if he/she were to be making a complaint.

SQL QUERIES

j. Count how many orders that are in the system:

This query will retrieve a figure that states the total number of orders currently in the system. The Orders table is the focus here.

Statement:

```
SELECT COUNT(*) FROM Orders;
```

Result:

COUNT(*)
5

Fig. 35

As seen in fig. 35, there are 5 orders in the system. The system here refers to the E-shop's database. The deliveries department can make use of this information to facilitate dispatch of the ordered products.

USE CASE DESIGN RATIONALE

The E-shop is a great example of the pivotal role of data in an online business. Customer information, products details and transaction records can shape the company's strategies, enable personalized services and targeted marketing.

- The design rationale is directed towards establishing clear relationships between entities. For instance, the decision to connect customers with orders and products was to aid seamless processing and personalized customers experiences (Nicola A. Amato, 2023).
- Normalization is important to ensure data integrity and efficiency by utilizing manageable tables and ensuring each table serves a specific purpose (Erki Eessaar, 2016).
- Decisions regarding data types, primary keys, foreign keys and constraints were made with a focus on effective performance. For example, the chosen data types ensured consistent and accurate data entry.
- By combining analytics and visual presentations, the system extracts meaningful insights from data sources.
- The use of Relational Database Management Systems (RDBMS) for the E-shop's database design gives it an organized structure that facilitates efficient business operations and enhances corporate business models (Date C.J., 2019).

CRITICAL ANALYSIS AND REFLECTION OF THE USE CASE

The E-shop's use case is a great basis to test out the implementation of ER diagram, database design and RDBMS. Nevertheless, I noticed there are some key things to note from the exploration of this use case:

- Normalization eliminates redundancy and enhances data integrity but while it is useful, there is a need for continuous evaluation between normalization and performance. There will be some instances where denormalization will significantly increase query speed but in this instance, the use of junction tables helped to establish the many-to-many relationships better.
- Regular review of the data types, primary keys and constraints choices is necessary to accommodate evolving business requirements and technological advances as they are key to the overall efficiency of the database.
- Identifying and addressing challenges during the design phase is essential and the effectiveness of the solutions should be periodically assessed. Continuous monitoring of system performance would help identify possible scalability challenges early.

RECOMMENDATION

Below are some suggestions that can potentially enhance the database usage and allow easy scalability implementations:

- Caching mechanism can be introduced to reduce database load for frequently accessed data.
- Implement indexing on frequently queried columns to improve query performance.
- Large tables can be partitioned to enhance data retrieval speed.
- Implement database replication to ensure availability and distribute the workload.
- Database sharding can be explored to distribute partitioned data across multiple servers for enhanced scalability.
- Upgrade hardwares to accommodate increased data and user loads.

CONCLUSION

A well-designed database is crucial for the E-shop's success as it ensures streamlined operations, data accuracy and adaptability to future business needs.

The recommendations given can ensure performance optimization and ensuring growing data volumes. Strategic design choices and the application of normalization principles and other key implementation decisions helped to create an efficient and purposeful database for the E-shop.

The design process and its outcomes underscores the critical role of effective database management in driving the e-commerce's platform functionality, scalability and overall success.

Fig. 36


Q & A SESSION
ANY QUESTIONS?

REFERENCES

Abraham Silberschatz, Henry Korth, S. Sudarshan, 2019. Database System Concepts. 7th edition. New York: McGraw-Hill.

Date C.J, 2019. Database Design and Relational Theory Normal Forms and All That Jazz. 2nd Edition. New York: Apress.

Erki Eessaar, 2016. The Database Normalization Theory and the Theory of Normalized Systems: Finding a Common Ground. Available at: https://www.researchgate.net/publication/297731569_The_Database_Normalization_Theory_and_the_Theory_of_Normalized_Systems_Finding_a_Common_Ground. Accessed on: 12th December, 2023.

Ignasio L. Bisso, 2022. What are the basic SQL queries? Available at: https://learnsql.com/blog/what-are-basic-SQL-queries/. Accessed on 11th December, 2023.

Knaflic, Cole Nussbaumer, 2015. Storytelling with data: A data visualization guide for business professionals. New Jersey: Wiley.

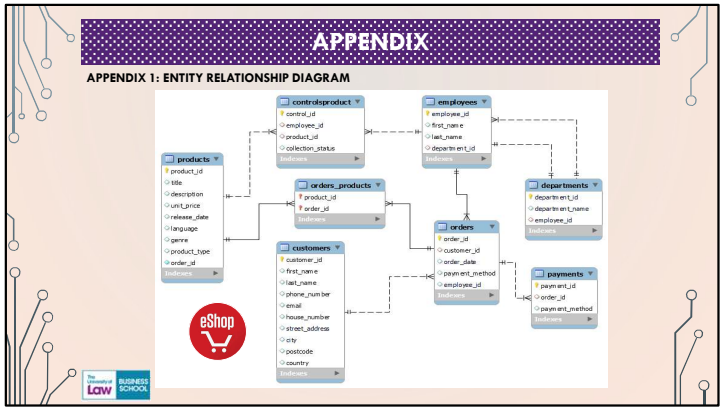
REFERENCES

Nicola A. Amato, 2023. Mastering database normalization: A comprehensive exploration of normal forms. Available at: https://www.researchgate.net/publication/374509386_Mastering_database_normalization_A_comprehensive_exploration_of_normal_forms. Accessed on: 9th December, 2023.

Ralph Kimball, Margy Ross, 2013. The data warehouse toolkit the definitive guide to dimensional modeling. Third edition. Indianapolis: John Wiley & Sons.

Stackshare, 2024. MySQL vs MySQL Workbench. Available at: https://stackshare.io/stackups/mysql-vs-mysql-workbench. Accessed on 11th December, 2023.

Thomas Connolly and Carolyn Begg, 2015. Database Systems: A Practical Approach to Design, Implementation, and Management, Global Edition. 6th edition. London: Pearson.



APPENDIX 2: SQL SCRIPTS

--Create a database

CREATE DATABASE virtualshop;

--Create the customers table

CREATE TABLE Customers (customer_id VARCHAR(50) PRIMARY KEY ,first_name VARCHAR(50),
last_name VARCHAR(50),
phone_number VARCHAR(20), email VARCHAR(100), house_number VARCHAR(10), street_address

VARCHAR(100), city VARCHAR(50), postcode VARCHAR(10), country VARCHAR(50));

--Using sample --data that meets specified criterion, populate the --table

INSERT INTO Customers (customer_id, first_name, last_name, phone_number, email, house_number, street_address, city, postcode, country)

VALUES

('C0001', 'Joy', 'Bob', '07065464324', 'joy@shopme.com', '22b', 'Bridge Avenue', 'Kent', '0921', 'UK'),

('C0002', 'Samuel', 'Mason', '07675467368', 'samuel@shopme.com', '87a', 'Apex Street', 'Bristol', '1290', 'UK'),

('C0003', 'Jane', 'George', '07568584373', 'jane@shopme.com', '91', 'Gate Road', 'Kent', '0831', 'UK'),

('C0004', 'Joshua', 'Ben', '07689706473', 'joshua@shopme.com', '897', 'Glory Lane', 'Manchester', '4576', 'UK'),

('C0005', 'Samantha', 'Grace', '07953626578', 'samantha@shopme.com', '56', 'Peace Street', 'Bristol','1378', 'UK');

--Products table will now be created with sample values.

```
CREATE TABLE Products (product_id VARCHAR(50) PRIMARY KEY, title VARCHAR(100), description TEXT, unit_price
DECIMAL(10, 2), release_date DATE, language VARCHAR(50), genre VARCHAR(50), product_type
VARCHAR(50));
INSERT INTO Products (product_id, title, description, unit_price, release_date, language, genre, product_type)
VALUES
('P0001', 'I Trend!', 'Classic Sci-Fi movie', 18.99, '2001-09-01', 'English', 'Science Fiction', 'Movie'),
('P0002', 'The Scroll', 'The Dystopian novel by George Orwell', 9.99, '1999-05-11', 'English', 'Dystopian', 'Book'),
('P0003', 'Trust', 'The Ben Graham album', 19.99, '2015-07-07', 'English', 'Pop', 'Music'),
('P0004', 'Inception', 'Mind-bending thriller', 14.99, '2016-10-07', 'English', 'Thriller', 'Movie'),
('P0005', 'The Hobbit', 'Fantasy novel by J.R.R. Tolkien', 24.99, '2011-09-20', 'English', 'Fantasy', 'Book');

--Orders and other tables are created similarly, defining constraints where necessary.
CREATE TABLE Orders (order_id VARCHAR(50) PRIMARY KEY, customer_id VARCHAR(50), product_id VARCHAR(50),
order_date DATE, payment_method VARCHAR(50),
FOREIGN KEY (customer_id) REFERENCES Customers(customer_id),
FOREIGN KEY (product_id) REFERENCES Products(product_id));
INSERT INTO Orders (order_id, customer_id, product_id, order_date, payment_method)
```

```
VALUES
('OR0001', 'C0001', 'P0001', '2022-11-05', 'Credit Card'),
('OR0002', 'C0002', 'P0002', '2022-10-20', 'Paypal'),
('OR0003', 'C0003', 'P0003', '2022-10-24', 'Credit Card'),
('OR0004', 'C0001', 'P0004', '2022-10-28', 'Cash'),
('OR0005', 'C0002', 'P0005', '2022-11-02', 'Loyalty Card');

CREATE TABLE Departments
(department_id VARCHAR(50) PRIMARY KEY, department_name VARCHAR(50));
INSERT INTO Departments (department_id, department_name)
VALUES
('DELVS', 'Deliveries'),
('ACCT', 'Accounting'),
('HR', 'Human Resources');

CREATE TABLE Employees
(employee_id VARCHAR(50) PRIMARY KEY, first_name VARCHAR(50), last_name VARCHAR(50), department_id
```

```
VARCHAR(50),
    FOREIGN KEY (department_id) REFERENCES Departments(department_id));
INSERT INTO Employees (employee_id, first_name, last_name, department_id)
VALUES
('EM001', 'Micah', 'Homes', 'DELVS'),
('EM002', 'Sarah', 'Williams', 'ACCT'),
('EM003', 'Chris', 'Buyer', 'HR'),
('EM004', 'Martha', 'Simpson', 'HR'),
('EM005', 'Fred', 'Davidson', 'DELVS');

--To establish the relationship between departments and employees, employee_id is needed.
ALTER TABLE Departments
ADD COLUMN employee_id VARCHAR(50);
ALTER TABLE Departments
ADD FOREIGN KEY (employee_id) REFERENCES Employees(employee_id);

CREATE TABLE Payments
```

(payment_id INT PRIMARY KEY AUTO_INCREMENT, order_id VARCHAR(50), payment_method VARCHAR(50),
FOREIGN KEY (order_id) REFERENCES Orders (order_id));
INSERT INTO Payments (payment_id, order_id, payment_method)
VALUES
('1', 'OR0001', 'Credit Card'),
('2', 'OR0002', 'Paypal'),
('3', 'OR0003', 'Credit Card'),
('4', 'OR0004', 'Cash'),
('5', 'OR0005', 'Loyalty Card');

--This is a junction table between employees and products table.
CREATE TABLE ControlsProduct
(control_id CHAR(50) PRIMARY KEY, employee_id VARCHAR(50), product_id VARCHAR(50),
collection_status VARCHAR(50),
FOREIGN KEY (employee_id) REFERENCES Employees (employee_id),
FOREIGN KEY (product_id) REFERENCES Products (product_id));
INSERT INTO ControlsProduct (control_id, employee_id, product_id, collection_status)

VALUES

('CP1', 'EM001', 'P0002', 'Collected'),

('CP2', 'EM001', 'P0004', 'Pending'),

('CP3', 'EM001', 'P0003', 'Dispatched'),

('CP4', 'EM001', 'P0001', 'Delivered'),

('CP5', 'EM001', 'P0005', 'Pending');

--SQL QUERIES

- *SELECT * FROM Customers WHERE city = 'Kent';*
- *SELECT * FROM Products WHERE genre = 'Fantasy';*
- *SELECT COUNT(*) FROM Customers WHERE city = 'Bristol';*
- *SELECT AVG(unit_price) FROM Products;*
- *SELECT * FROM Orders;*
- *SELECT * FROM Orders*

JOIN Products ON Orders.product_id = Products.product_id

WHERE Products.product_type = 'Book' AND Products.description LIKE '%the%';

- *SELECT * FROM Payments*

JOIN Orders ON Payments.order_id = Orders.order_id

JOIN Products ON Orders.product_id = Products.product_id

WHERE Payments.payment_method = 'Credit Card' AND Products.product_type = 'Music';

- *SELECT COUNT(*) FROM Employees*

JOIN Orders ON Employees.employee_id = Orders.employee_id

JOIN Products ON Orders.product_id = Products.product_id

WHERE Products.product_type = 'Music';

- *SELECT COUNT(*) FROM Employees WHERE first_name LIKE 'S%';*
- *SELECT COUNT(*) FROM Orders;*