Introduction to Web and Database Concepts

ACCA4023_LON
Design and Implementation of a Relational Database System

Rosa Lucena Saladie
2337012

TABLE OF CONTENTS

1. PROJECT SUMMARY

CraftGem is a fast-growing retailer of jewellery crafted by hand yet relies on manual processes to handle main operations and core business functions. With the company's growth, manually managing inventory stock, customer data, and orders has become obsolete, resulting in delay errors and redundancies. The director noted that the approach to be taken in addressing this scenario and automating the main business processes is to implement a database management system, such as Microsoft Access.

Objective

This Microsoft Access database is a relational management system.[1] Designed to structure and manage inventory, restock requests, customer data, supplier data, revenue and expenses, and employee performance, as well as to visualise reports for wise decision-making. CraftGem can replace the manual record-keeping process with a structured database, enhancing accuracy, productivity, and overall business performance.

1.1. Features

- **Inventory Tracking**: Keep track of products, stock levels, suppliers, and stocking when necessary.
- **Order Processing:** Monitor customer orders, payments, and status efficiently.
- **Customer Management**: We store and analyse key customer information, including contact details, order history, and loyalty points. This allows us to understand customer profiles and identify top customers to create tailored marketing campaigns in the future.
- **Employee Performance:** Track and evaluate employee performance to calculate bonuses based on sales achievements.
- **Reporting & Analytics:** sales, stock levels, and customer activity overview.

1.2. Database Design and Integrity

The attached Entity-Relationship Diagram (ERD) represents the relationships between essential business entities, ensuring a well-structured and logically organised database. This design facilitates efficient data management, boosts performance, and maintains data consistency.

By using Declarative Referential Integrity,[2] This DB implements strict data governance through the following constraints:

- Primary Keys (PK): Identify each record in key Entities (e.g., CustomerID, OrderID, ProductID) to prevent duplication.
- Foreign Keys: Define relationships between entities, ensuring accurate associations between customers, orders, products and suppliers.
- UNIQUE Constraints: Prevent duplicates for attributes such as customer and employee emails and phone numbers.
- Data Validation Rules and input masks: Limit invalid input, maintain data accuracy and ensure that there is no inconsistency in the data

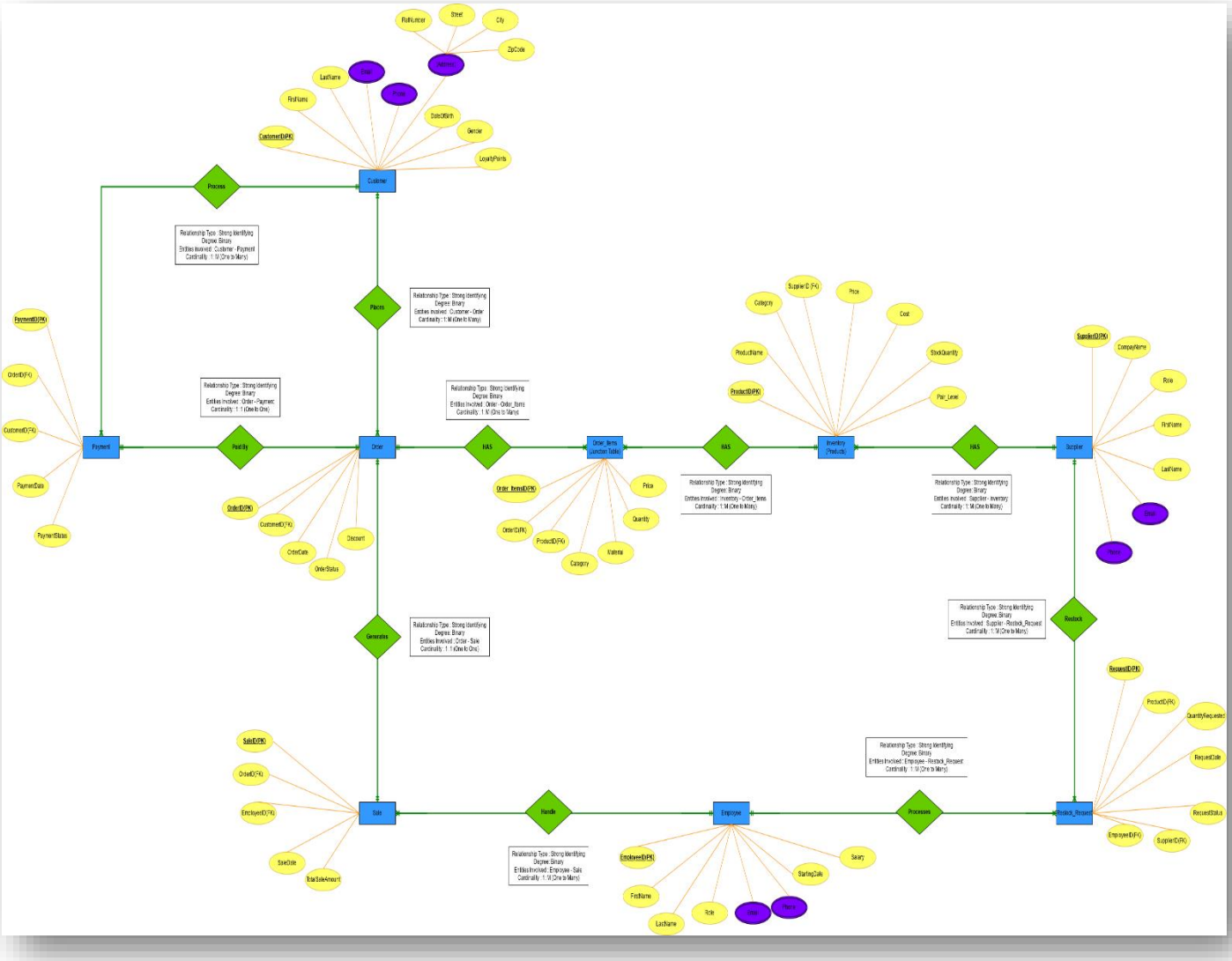
This structured approach enables seamless data retrieval and reduces data redundancy. It supports business activities as inventory tracking, customer management, financial reporting, and employee performance evaluations. Additionally, it lays the foundation for analytical insights, allowing for data-driven decision-making and tailored marketing strategies.

2. ENTITY RELATIONSHIP DIAGRAM (ERD):

The Entity-Relationship (ER) model is a foundation of the conceptual view within the logical data model. Focusing on entities, attributes, and their relationships, it offers an abstract and comprehensive representation of the database structure [3].

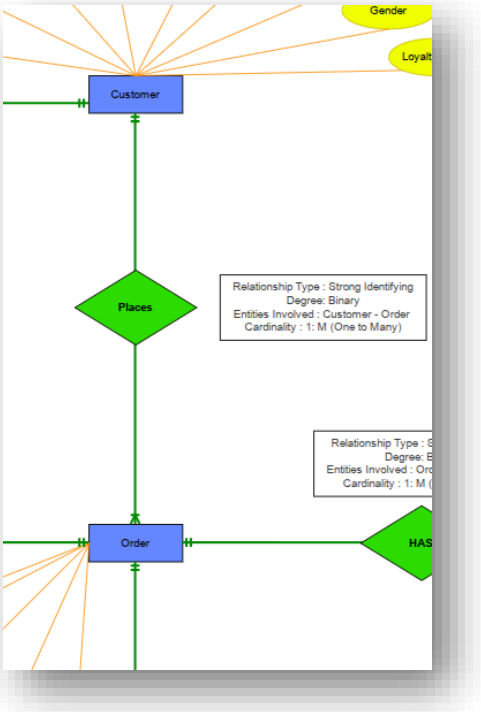
The ER diagram, introduced by Peter Chen in 1976, [4] revolutionised the way databases are structured and understood. Chen provided a conceptual, visual representation of how data is structured and related within a database. Before his work, database modelling lacked a standard, intuitive framework, making it challenging to design and communicate complex relationships effectively.

2.1. OVERVIEW OF THE CREAFTGEM ERD



2.2. ENTITIES RELATIONSHIPS DESCRIPTION

2.2.1. CUSTOMER – ORDER

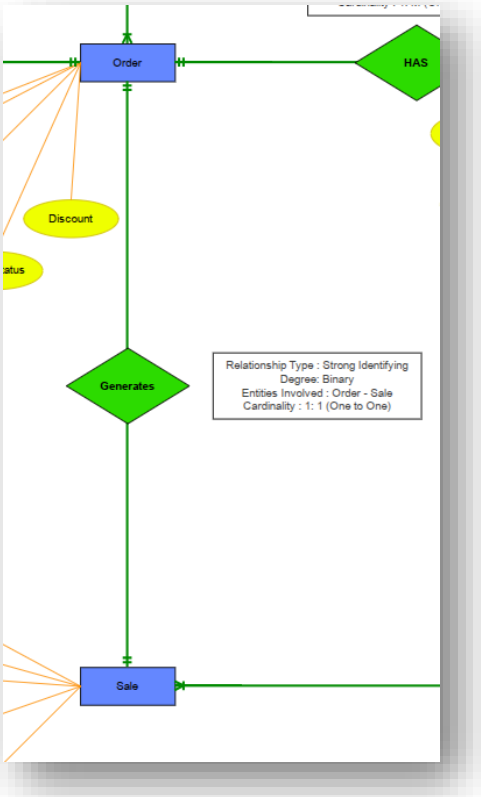


The Customer and Order entities share a strong, identifying relationship with a binary degree between them.

The cardinality of this relationship is one-to-many (1:M) with a mandatory character because one customer can place one or multiple orders, but each order must be associated with exactly one customer.[5], [6]

It is labelled as "places", indicating that a Customer places an Order. The connection is enforced through CustomerID, which serves as a foreign key (FK) in the Order table, referencing the primary key (PK) in the Customer table. This ensures referential integrity, preventing an order from existing without a valid associated customer.

2.2.2. ORDER – SALES



The Order and Sales entities share a strong identifying relationship with a binary degree between them.

The cardinality of this relationship is one-to-one (1:1) with a mandatory character, meaning that each order must generate exactly one sale and correspond to exactly one order.

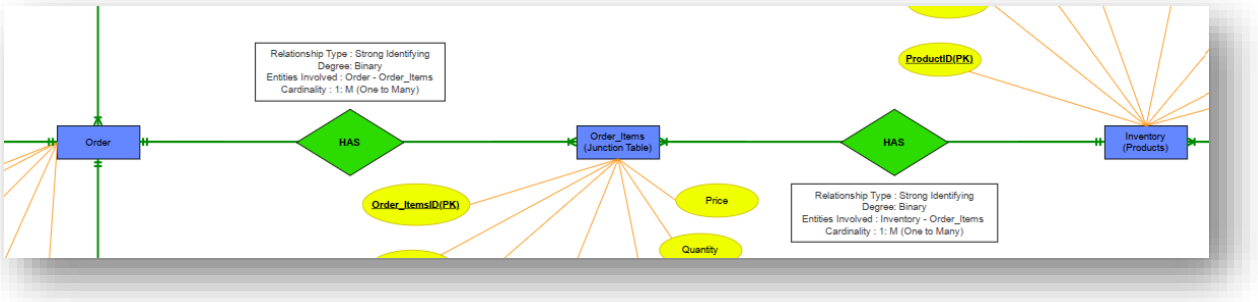
This relationship is labelled "generates", indicating that an Order generates a Sale. The connection is enforced by the foreign key OrderID in the Sales table, which references the Order table's primary key.

2.2.3. ORDER – INVENTORY (PRODUCTS)

The Order and Inventory (Products) entities share a strong identifying binary relationship. The cardinality of this relation is many-to-many because one order can have different products, and various products can be in one order. This relationship is normalised by introducing a junction table called Order_Items. This table resolves the many-to-many relationship by breaking it into two one-to-many relationships with a mandatory character.

- **Order - Order_Items:** A single order can contain multiple order items. The Order_Items table includes a foreign key, OrderID, referencing the Order entity. The cardinality one-to-many relationship ensures that an order can have multiple products.
- **Inventory (Products) – Order_Items**: A single product from the inventory can be part of multiple order items. The Order_Items table includes a foreign key, ProductID, referencing the Inventory (Products) entity. The cardinality one-to-many relationship ensures that a product can be included in multiple orders.

This overall relationship is labelled as "has," indicating that an order has order items, which correspond to products from the inventory. Foreign keys in the Order_Items table maintain referential integrity, ensuring that orders reference valid products and adhere to normalisation standards by eliminating a direct many-to-many relationship.

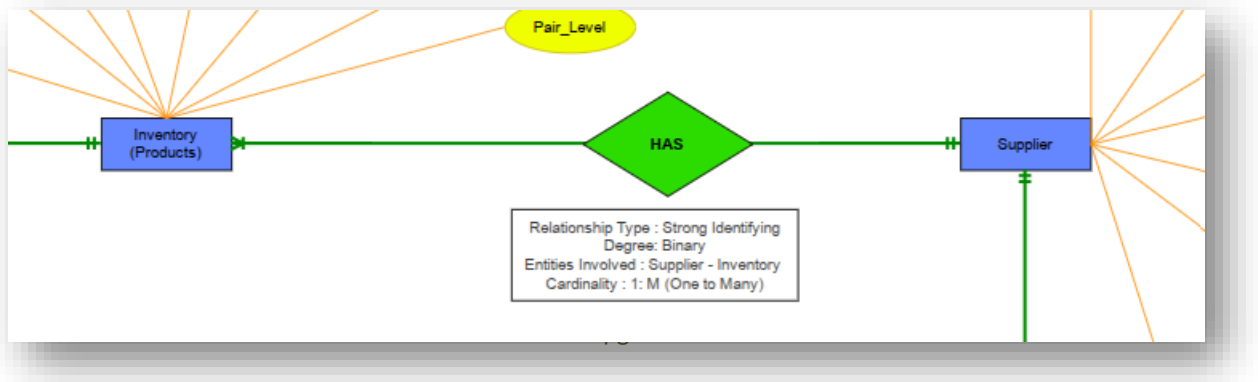


2.2.4. SUPPLIER – INVENTORY

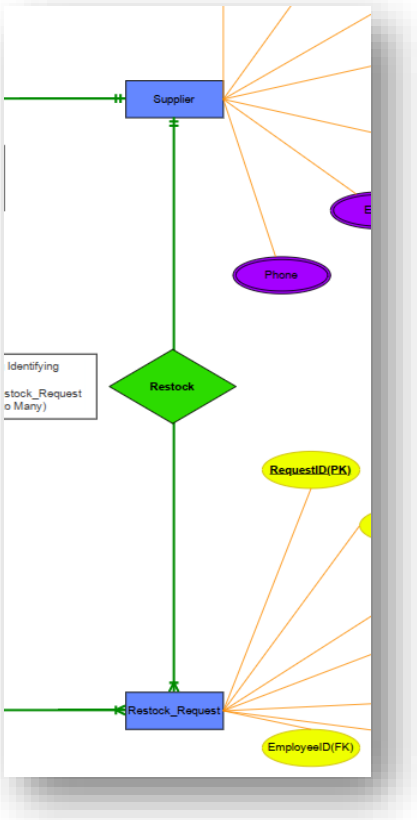
The Supplier and Inventory (Products) entities share a strong, binary relationship in degree.

The cardinality is one-to-many (1:M), meaning that one supplier can provide multiple products, but each is supplied by exactly one supplier. This relationship is mandatory on the product side, ensuring every product in the inventory is linked to a valid supplier.

This relationship is labelled as "supplies", highlighting that a supplier provides one or more products. In this setup, the Inventory (Products) entity includes a foreign key, SupplierID, which references the primary key in the Supplier entity. This linkage ensures that every product is associated with a valid supplier, maintaining data integrity and supporting efficient tracking of product sourcing within the database.



2.2.5. SUPPLIER – RESTOCK_REQUEST



The Supplier and Restock_Request entities share a strong identifying relationship with a binary degree involving both entities.

The cardinality of this relationship is one–to–many (1:M), where each supplier can handle different restock_requests, but each restock_request is associated with a specific supplier.

This relationship is labelled as "restock", highlighting that suppliers attend the restock requests placed by the employees.

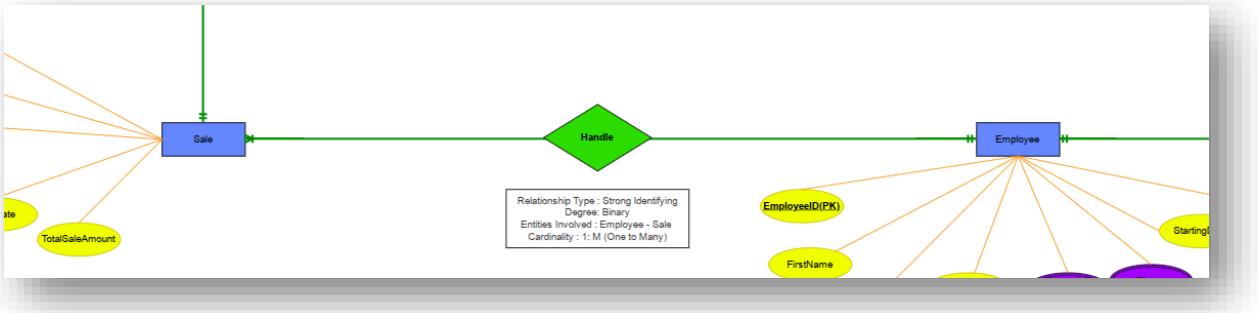
SupplierID is stored as a foreign key in the Restock_Request table. This guarantees that each restock request is linked to a valid supplier while keeping RequestID as the primary key.

2.2.6. EMPLOYEE – SALES

The Employee and Sales entities share a strong identifying relationship that is binary in degree because the foreign key EmployeeID in the Sales entity is not part of its primary key, meaning that a sale can exist independently of the relationship.

The cardinality of this relationship is one-to-many (1:M). Each sale must be handled by exactly one employee, and each employee can handle multiple sales.

This relationship is labelled as "handles", indicating that an employee is responsible for processing sales transactions. The Sales table includes a foreign key, EmployeeID, which references the primary key in the Employee table. This foreign key constraint ensures that every sale is linked to a valid employee.



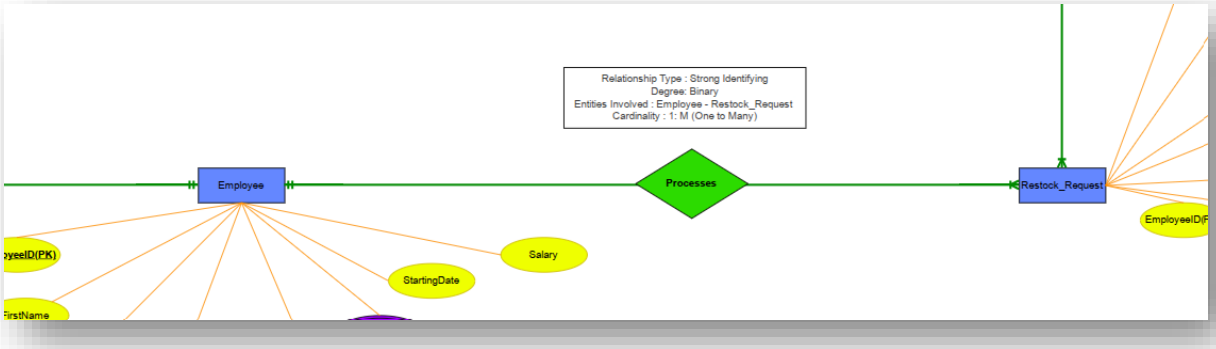
2.2.7. EMPLOYEE – RESTOCK_REQUEST

The Employee and Restock_Request entities share a strong identifying relationship with a binary degree, involving both the Employee and Restock_Request entities.

The cardinality is one-to-many (1:M) because one employee can process multiple restock requests, but each restock request must be associated with one employee. This relationship will not be mandatory because, in the future, the company wants to automate the restock_request without the need for the employee to be involved.

This relationship is labelled as "processes", indicating that employees handle restock requests when applicable.

The Restock_Request table includes a foreign key, EmployeeID, referencing the Employee entity. This maintains referential integrity, ensuring that when an employee is assigned to a request, they are valid within the system while allowing automated requests to exist independently.

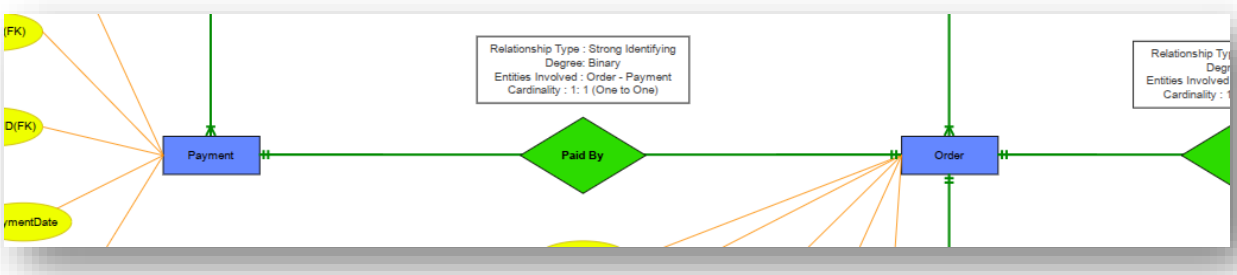


2.2.8. ORDER – PAYMENT

The Order and Payment entities share a strong identifying relationship with a binary degree.

The cardinality of this relationship is one-to-one (1:1) with a mandatory character in both sides, meaning that each order is associated with exactly one payment**,** and each payment corresponds to exactly one order. To maintain referential integrity**,** the OrderID is stored as a foreign key in the Payment table, ensuring that each payment is associated with an existing order.

At this stage, the design only considers full payments**,** meaning each order is paid through a single payment. Future considerations could allow for partial payments (i.e., multiple payments for one order), changing the relationship to one-to-many**.** This relationship is labelled as "paid by"**,** indicating that a payment pays for an order.

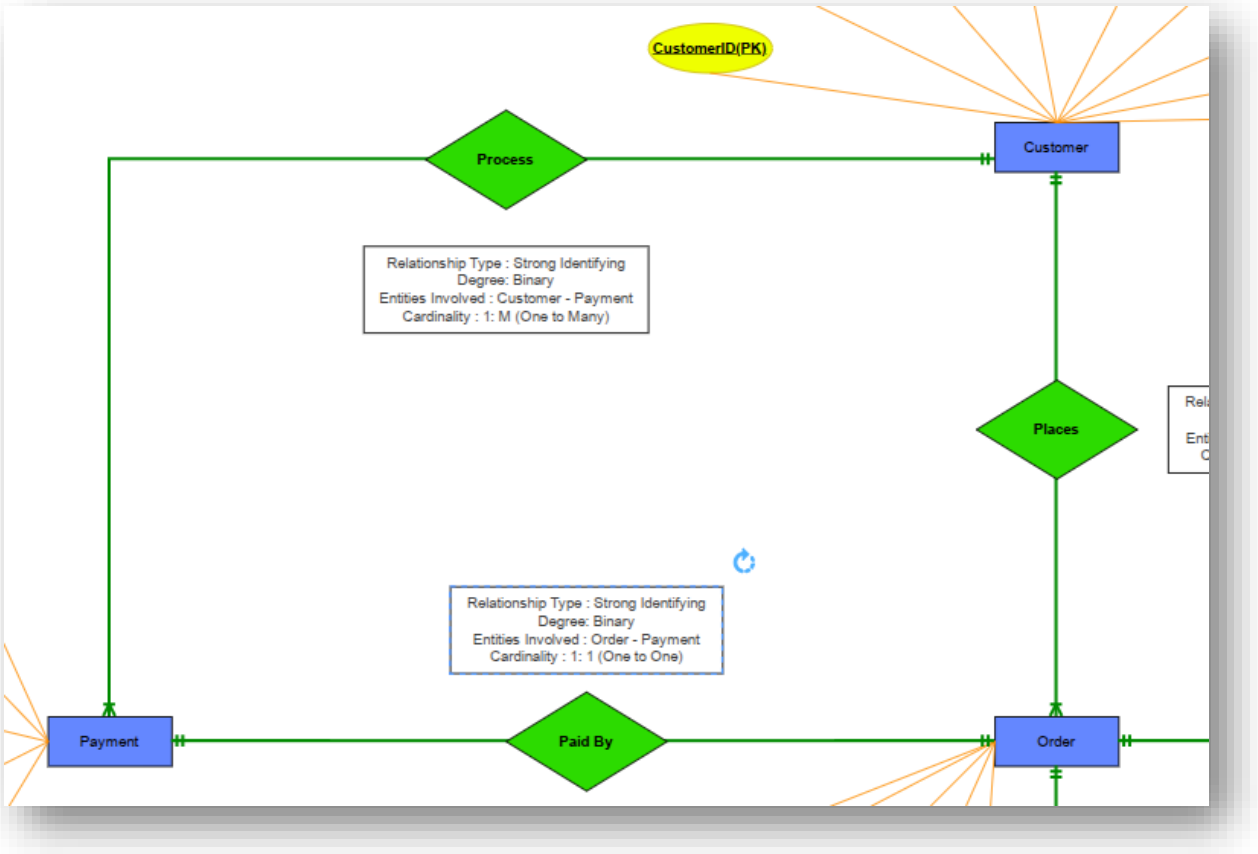


2.2.9. CUSTOMER - PAYEMENT

The Customer and Payment entities share a strong identifying relationship with a binary degree.

The cardinality of this relationship is one-to-many (1:M), meaning a single customer can make multiple payments, but each payment is associated with exactly one customer.

This relationship is labelled as "process," indicating that a customer makes payments. The CustomerID foreign key in the Payment table ensures that each payment is associated with a valid customer, ensuring data integrity and proper tracking of customer payments in the system.



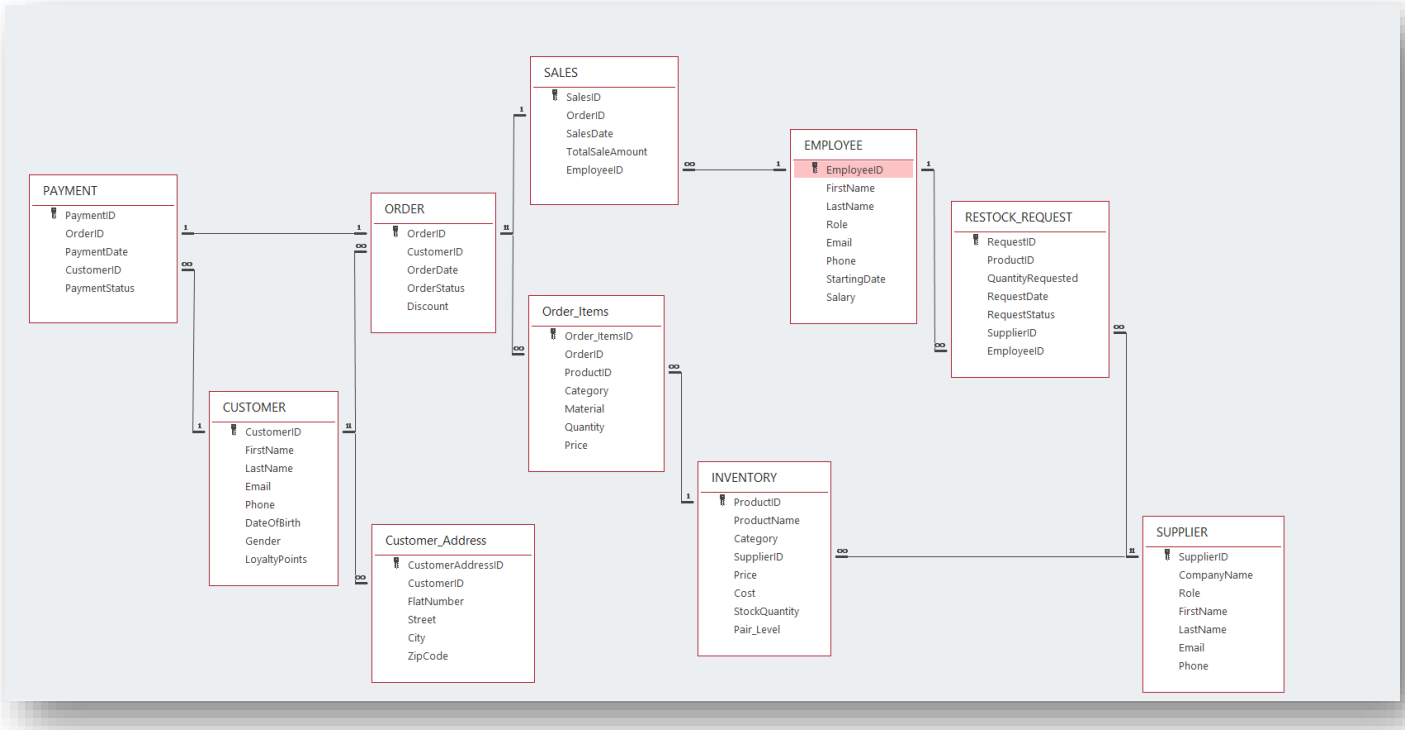
3. LOGICAL DESIGN

By the end of this phase, I had developed a solid, logical design for the database, ensuring that the structure was well-organized, adhered to best practices in database design, and was ready to be implemented in the following stages of development.

- **Database Creation:** I started this phase by developing a new database using Microsoft Access, functioning as the foundation for all business data storage. Part of this involved setting up the database environment and getting it ready to make tables and relationships
- **Table Design:** I created the tables corresponding to each Entity in the system (e.g., Customer, Order, Payment, Product). I defined the respective tables or entities, the relevant attributes(fields), and the most appropriate data type for those fields. This means that data must be stored properly. For example, IDs must be stored as integers, Names as text, Dates as dates, and Amounts in currency.
- **Establishing Relationships:** I identified how the tables would relate to one another to ensure that they would be linked appropriately, and that the database would maintain referential integrity.
- **Normalisation to 3NF:** I applied the Third Normal Form (3NF) [7], to improve consistency and performance. This process includes removing redundant data and ensuring that all the non-key attributes in the tables are entirely dependent on the primary key, effectively removing any partial or transitive dependencies. The homogenisation process enabled a more efficient architecture, minimising excessive redundancy and governing long-term storage management.

By the conclusion of this phase, I had constructed a database with a well-organized structure and adhered to optimal database design best practices.

3.1. OVERVIEW OF THE CREATING ACCESS DATABASE



3.2. TABLES DESIGN

3.2.1. CUSTOMER TABLE

The Customer table stores information about individual customers. The attributes of this entity are as follows:

- CustomerID (PK): A unique identifier for each customer.
- FirstName: The customer's first name.
- LastName: The customer's last name.
- Email: A unique, multivalued attribute that can store different email addresses per customer.
- Phone: A multivalued attribute that can store different phone numbers per customer.
- Address: A Composite attribute consisting of multiple components such as Flat Number, Street, City, Zip Code, etc.
- DateOfBirth: The customer's date of birth.
- Gender: The gender of the customer.
- LoyaltyPoints: The number of loyalty points accumulated by the customer.

The Customer Table has been normalised into Third Normal Form (3NF) as they are atomic attributes. We have a unique primary key for every record called CustomerID, and everything that was not a key depended entirely on CustomerID. No attribute depends on a non-key attribute—it is determined only by CustomerID.

- **Email and Phone:** Since a customer can have multiple emails and phone numbers, creating separate tables (e.g., CustomerEmail and CustomerPhone) with CustomerID as a foreign key would be the best practice. These tables would enforce unique constraints on the Email and Phone fields. They will prevent duplicate entries for a single customer, ensuring that each email and phone number is unique per customer.
- **Address:** Since Address is a composite attribute [8](Street, city, and postal code), the best design would be a separate Customer_Address table. This is even more useful for customers since they can have multiple addresses, which normalises the data even more. The relationship between Customer and Customer_Address will be (1:M) One to Many.
- **Current Project Scenario:** For this project, I have enforced unique constraint for Email & Phone. But, for my db. solution, I am going to consider that only one unique email and one unique phone number exist for a customer, so Email and Phone will remain as simple attributes in the Customer table. For developing customer address as it is composite I developed separate table where CustomerID is foreign key.

CUSTOMER

Field Name	Data Type	Description (Optional)
CustomerID	AutoNumber	A unique identifier for each customer
FirstName	Short Text	The customer's first name
LastName	Short Text	The customer's last name
Email	Short Text	A unique multivalued attribute that can store different email addresses per customer
Phone	Short Text	A unique multivalued attribute that can store different phone numbers for each customer
DateOfBirth	Date/Time	The customer date of birth
Gender	Short Text	The gender of the customer
LoyaltyPoints	Number	The number of loyalty points accumulated by the customer

Customer_Address

Field Name	Data Type	Des
CustomerAddressID	AutoNumber	A unique identifier for each customer_Address
CustomerID	Number	Foreign Key to link Customer and Customer_Address
FlatNumber	Short Text	Customer flat number
Street	Short Text	Street where customer lives
City	Short Text	City where customert lives
ZipCode	Short Text	Customer zip code

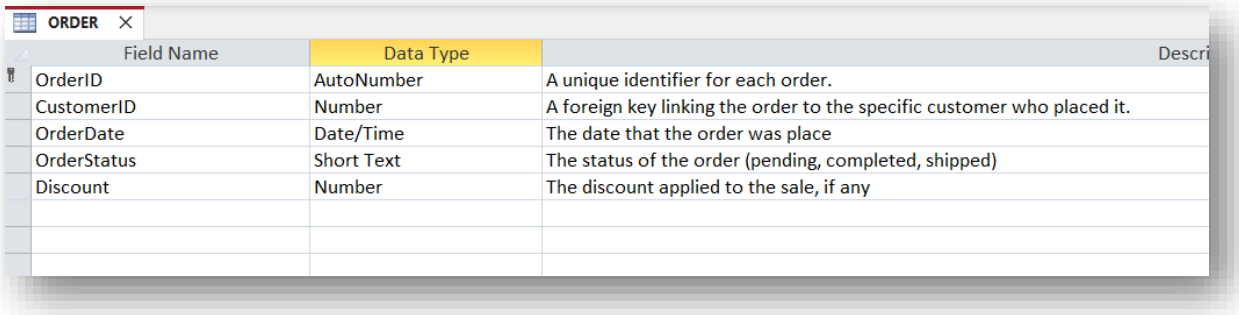
3.2.2. ORDER TABLE

The Order entity stores information about individual orders placed by customers. The attributes of this entity are as follows:

- OrderID (PK): A unique identifier for each order.
- CustomerID (FK): A foreign key linking the order to the specific customer who placed it.
- OrderDate: The date the order was placed.
- OrderStatus: The status of the order (pending, completed, shipped)
- Discount: The discount applied to the sale, if any

The Order table is in Third Normal Form (3NF) because every attribute is atomic, and each record has a unique identifier (OrderID). All non-key attributes depend entirely on OrderID, ensuring no partial or transitive dependencies exist.

- **OrderStatus**: While OrderStatus is a relatively simple attribute, best practices would be to store all possible order statuses in a separate table (e.g., OrderStatusType) and reference it via a foreign key. This approach gives flexibility and prevents inconsistencies with status values.
- **Discount:** Similarly, if discounts are applied from a predefined set of different discounts, the Discount attribute could be normalised into a separate table (e.g., DiscountType). The main table would then reference this table with a foreign key, helping to maintain consistency in discount values.
- **Current Project scenario:** For this project, however, OrderStatus and Discount will be represented as simple attributes within the same Order table. In this case, CustomerID will be implemented as a **foreign key**, ensuring that each child table (Order) record is associated with a valid parent table (Customer) record.


ORDER

Field Name	Data Type	Descri
OrderID	AutoNumber	A unique identifier for each order.
CustomerID	Number	A foreign key linking the order to the specific customer who placed it.
OrderDate	Date/Time	The date that the order was place
OrderStatus	Short Text	The status of the order (pending, completed, shipped)
Discount	Number	The discount applied to the sale, if any

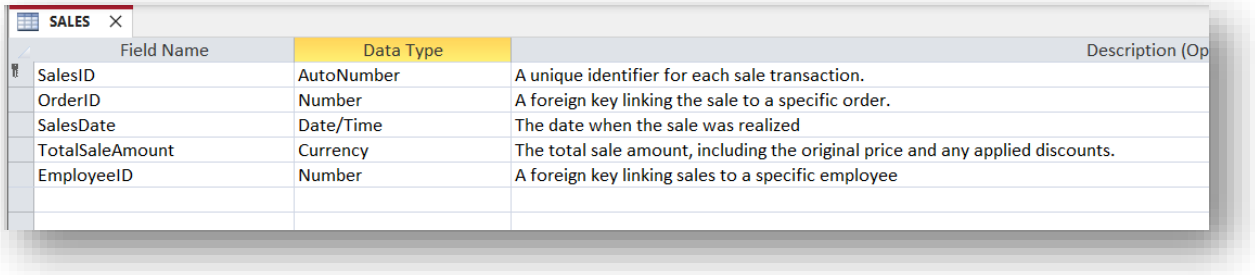
3.2.3. SALES TABLE

The Sales entity stores information about individual sales transactions linked to orders. The attributes of this entity are as follows:

- SaleID (PK): A unique identifier for each sale transaction.
- OrderID (FK): A foreign key linking the sale to a specific order.
- SaleDate: The date when the sale was made.
- TotalSaleAmount: The total sale amount, including the original price and any applied discounts.
- EmployeeID (FK): A foreign key linking sales to a specific employee

The table is in Third Normal Form (3NF) because each attribute is atomic, and each record has a unique identifier (SaleID). All non-key attributes depend entirely on SaleID, and there are no partial or transitive dependencies—all attributes depend directly on SaleID.

- The foreign key OrderID enforces the connection, ensuring referential integrity and maintaining the one-to-one correspondence between orders and sales.



SALES

Field Name	Data Type	Description (Op
SalesID	AutoNumber	A unique identifier for each sale transaction.
OrderID	Number	A foreign key linking the sale to a specific order.
SalesDate	Date/Time	The date when the sale was realized
TotalSaleAmount	Currency	The total sale amount, including the original price and any applied discounts.
EmployeeID	Number	A foreign key linking sales to a specific employee

3.2.4. ORDER_ITEMS TABLE

The Order_Items entity stores information about individual items in an order. Each record represents a specific product within an order. The attributes of this entity are as follows:

- Order_ItemID (PK): A unique identifier for each order item.
- OrderID (FK): A foreign key linking the order item to a specific order in the Order table.
- ProductID (FK): A foreign key linking the order item to a specific product in the Product table.
- Category: Specifies the category of the product, allowing for classification and filtering.
- Material: Defines the material of the product, providing additional specificity.
- Quantity: The number of units of the product ordered.
- Price: The price of a single unit of the product at the time of purchase.

The Order_items table is in Third Normal Form (3NF) because each attribute is atomic, and each record has a unique identifier attribute (Order_ItemID). All non-key attributes depend entirely on Order_ItemID, and there are no partial or transitive dependencies—all attributes depend directly on Order_ItemID.

- While it may seem logical to include Subtotal as a derived attribute (calculated as Quantity × Price), best practices dictate that it should not be stored in the database but calculated dynamically when needed.
- The foreign key OrderID enforces the connection, which ensures referential integrity and maintains the one-to-many correspondence between orders and other_items. The Order_Items table also includes a foreign key, ProductID, referencing the Inventory (Products) entity.

Order_Items

Field Name	Data Type	Description (O
Order_ItemsID	AutoNumber	A unique identifier for each order item.
OrderID	Number	A foreign key linking the order item to a specific order in the Order table.
ProductID	Number	A foreign key linking the order item to a specific product in the Product table.
Category	Short Text	Specifies the category of the product, allowing for classification and filtering.
Material	Short Text	Defines the material of the product, providing additional specificity.
Quantity	Number	The number of units of the product ordered.
Price	Currency	The price of a single unit of the product at the time of purchase.

3.2.5. INVENTORY (PRODUCT)TABLE

The Inventory (Products) table stores information about products available in stock. It helps track product details, stock levels, and supplier relationships. The attributes of this entity are as follows:

- ProductID (PK): A unique identifier for each product.
- ProductName: The name of the product.
- Category: The category to which the product belongs, which is useful for classification and filtering.
- SupplierID (FK): A foreign key linking the product to its supplier in the Suppliers table.
- Price: The selling price of a single unit of the product.
- Cost: The cost of acquiring or producing a single unit of the product.
- StockQuantity: The current number of units available in stock.
- Pair_Level: The minimum stock level at which a product should be reordered to prevent stockouts

The Inventory table is in Third Normal Form (3NF) because each attribute is atomic, and each record has a unique identifier (StockQuantityID). All non-key attributes depend entirely on StockQuantityID, and there are no partial or transitive dependencies—all attributes depend directly on StockQuantityID.

- **The StockQuantity:** does not track stock movements (e.g., purchases, restocking, or sales); a more detailed solution would be introducing a StockTransactions table. This table would include attributes such as TransactiProductID (FK), TransactionType (e.g., Sale, Restock), QuantityChanged, and TransactionDate to track inventory fluctuations over time effectively.
- This table includes a foreign key, SupplierID, which references the primary key in the Supplier entity. This linkage ensures that every product is associated with a valid supplier, maintaining data integrity and supporting transparent tracking of product sourcing within the database.
- **Current Project scenario:** Given this project's current scenario and requirements, I will keep StockQuantity as a simple attribute in the Inventory (Products) entity. This approach simplifies the design while still meeting the project's needs.

INVENTORY

Field Name	Data Type	Description (Optional)
ProductID	AutoNumber	A unique identifier for each product
ProductName	Short Text	The name of the product
Category	Short Text	The category to which the product belongs, which is useful for classification and filtering.
SupplierID	Number	A foreign key linking the product to its supplier in the Suppliers table.
Price	Currency	The selling price of a single unit of the product.
Cost	Currency	The cost of acquiring or producing a single unit of the product.
StockQuantity	Number	The current number of units available in stock.
Pair_Level	Number	The minimum stock level at which a product should be reordered to prevent stockouts

3.2.6. SUPPLIER TABLE

The supplier entity stores information about suppliers that provide products. Its attributes are as follows:

- SupplierID (PK): A unique identifier for each supplier.
- CompanyName: Name of the Company
- Role: The Role of the employee
- FirstName: The First name of the supplier company.
- LastName: The LastName of the supplier company
- Email: A unique, multivalued attribute that can store one email address per customer.
- Phone: A unique, multivalued attribute that can store one phone number per customer

The Supplier Table is in Third Normal Form (3NF) because each attribute is atomic, and each record has a unique identifier (SupplierID). All non-key attributes depend entirely on SupplierID, and there are no partial or transitive dependencies—all attributes depend directly on SupplierID.

- **Email and Phone:** I will enforce a unique constraint for Email and Phone attributes. However, I will assume that each customer has only one unique email and one unique phone number. The database will maintain Email and Phone as simple attributes within the Customer table.

SUPPLIER

Field Name	Data Type	Description (Optional)
SupplierID	AutoNumber	A unique identifier for each supplier
CompanyName	Short Text	Name of the company
Role	Short Text	Role of the employee
FirstName	Short Text	First name of supplier contact
LastName	Short Text	Last name of supplier contact
Email	Short Text	A unique, multivalued attribute that can store one email address per customer.
Phone	Short Text	A unique, A multivalued attribute that can store one phone number per customer

3.2.7. EMPLOYEE TABLE

The Employee entity captures details about each employee in the organisation. The attributes include:

- EmployeeID (PK): A unique identifier for each employee.
- FirstName: The employee's first name.
- LastName: The employee's last name.
- Role: The position or job title held by the employee.
- Email: A multivalued attribute, allowing an employee multiple email address.
- Phone: A multivalued attribute, enabling the storage of multiple phone numbers per employee.
- StartingDate: The date the employee was hired.
- Salary: The employee's salary.

The Table is in Third Normal Form (3NF) because each attribute is atomic, and each record has a unique identifier (EmployeeID). All non-key attributes depend entirely on EmployeeID, and there are no partial or transitive dependencies—all attributes depend directly on EmployeeID.

- **Email and Phone:** I will enforce a unique constraint for Email and Phone attributes for this project. However, I will assume that each customer has only one unique email and phone number. The database will maintain Email and Phone as simple attributes within the Employee table
- **Role:** The best approach would be to create a separate EmployeeRoles table, anticipating the creation of different roles for the company. This table would store standardised role definitions, reducing redundancy and ensuring consistency across the database. Based on the requirements for this project, I will consider role as a simple attribute in the same Employee table.

EMPLOYEE

Field Name	Data Type	Description (Optional)
EmployeeID	AutoNumber	A unique identifier for each employee.
FirstName	Short Text	The employee's first name
LastName	Short Text	The employee's last name
Role	Short Text	The position or job title held by the employee.
Email	Short Text	A multivalued attribute, allowing an employee multiple email address.
Phone	Short Text	A multivalued attribute, enabling the storage of multiple phone numbers per employee.
StartingDate	Date/Time	The date the employee was hired.
Salary	Currency	The employee's salary.

3.2.8. RESTOCK_REQUEST TABLE

The Restock_Request entity records requests to replenish inventory, capturing essential details for tracking and managing stock levels. Each record represents a specific request and includes the following attributes:

- RequestID (PK): A unique identifier for each restock request.
- ProductID (FK): A foreign key linking the request to a specific product in the Inventory (Products) table.
- QuantityRequested: The number of units requested for restocking.
- RequestDate: The date when the restock request was made.
- RequestStatus: The status of the request (e.g., pending, approved, completed, or rejected).
- SupplierID (FK): A foreign key linking the request to a specific supplier who will fulfil the request.
- EmployerID (FK): A foreign key linking the request to the employee (or employer) who initiated the request.

Each attribute is atomic, and the unique identifier (RequestID) ensures that all non-key attributes depend directly on it. This satisfies the Third Normal Form (3NF), as there are no partial or transitive dependencies.

- SupplierID and EmployeeID are stored as foreign keys. This guarantees that each restock request is linked to a valid supplier and a specific employee while keeping RequestID as the primary key.

RESTOCK_REQUEST

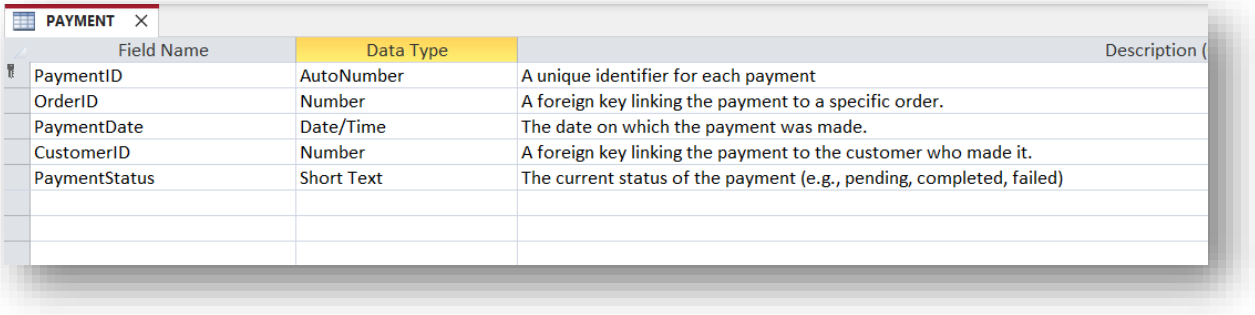
Field Name	Data Type	Description (Optional)
RequestID	AutoNumber	A unique identifier for each restock request
ProductID	Number	A foreign key linking the request to a specific product in the Inventory (Products) table.
QuantityRequested	Number	The number of units requested for restocking.
RequestDate	Date/Time	The date when the restock request was made.
RequestStatus	Short Text	The status of the request (e.g., pending, approved, completed, or rejected).
SupplierID	Number	A foreign key linking the request to a specific supplier who will fulfil the request.
EmployeeID	Number	A foreign key linking the request to the employee (or employer) who initiated the request.

3.2.9. PAYMENT TABLE

The Payment entity captures all details related to the payment transactions for orders. Each record in this entity represents a specific payment made by a customer for an order. The attributes include:

- PaymentID (PK): A unique identifier for each payment.
- OrderID (FK): A foreign key linking the payment to a specific order.
- PaymentDate: The date on which the payment was made.
- CustomerID (FK): A foreign key linking the payment to the customer who made it.
- PaymentStatus: The current status of the payment (e.g., pending, completed, failed)

The Payment table is in 3NF because each attribute contains a single value, it is atomic, and there are no partial or transitive dependencies: Every non-key attribute (OrderID, PaymentDate, CustomerID, and PaymentStatus) depends directly on the primary key (PaymentID), which ensures there are no partial or transitive dependencies.



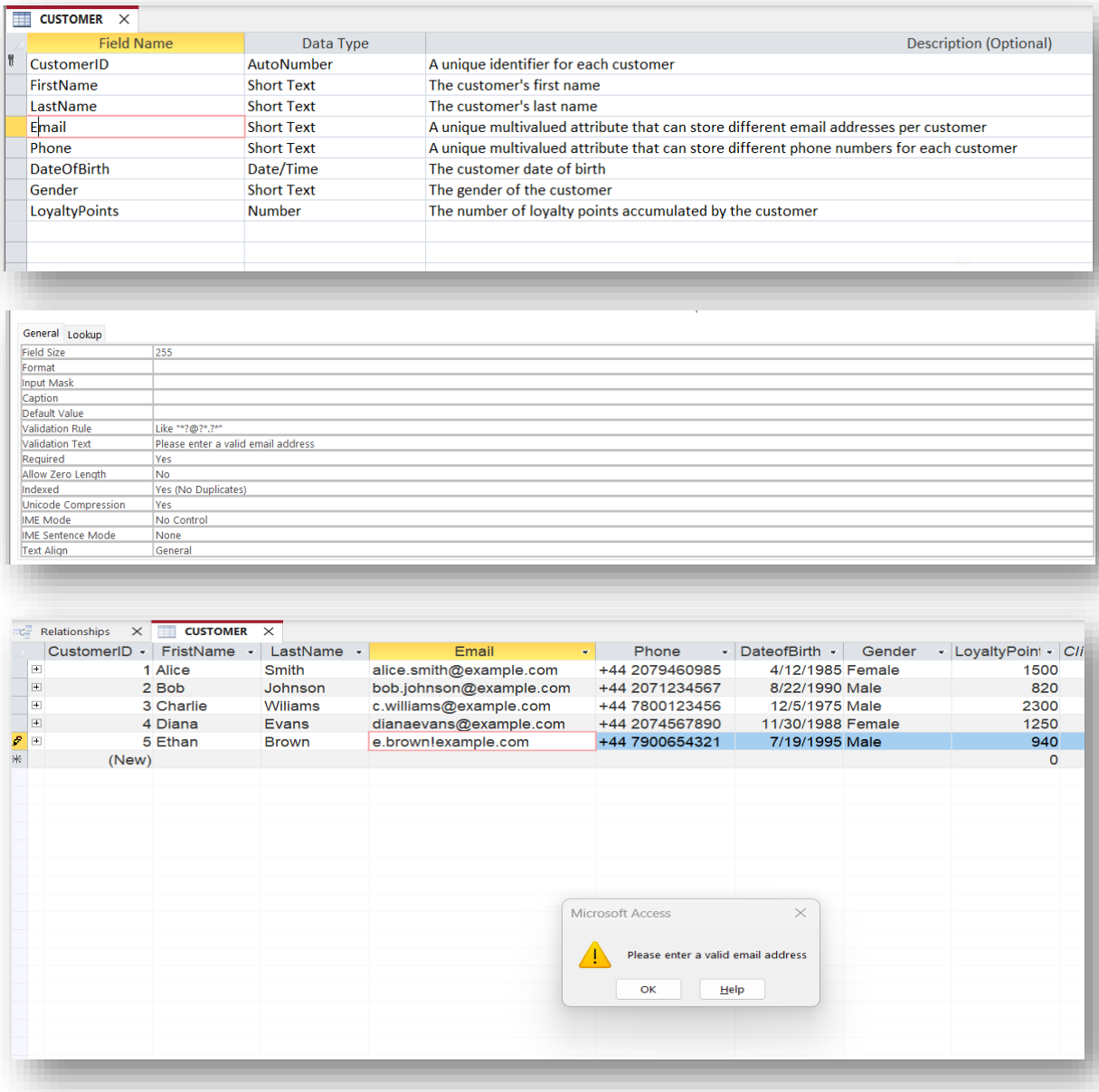
PAYMENT

Field Name	Data Type	Description (
PaymentID	AutoNumber	A unique identifier for each payment
OrderID	Number	A foreign key linking the payment to a specific order.
PaymentDate	Date/Time	The date on which the payment was made.
CustomerID	Number	A foreign key linking the payment to the customer who made it.
PaymentStatus	Short Text	The current status of the payment (e.g., pending, completed, failed)

3.3. TESTING AND VALIDATION

3.3.1. VALIDATION OF EMAIL WITH A VALIDATION RULE

In Microsoft Access, I validate email inputs using a validation rule (Like "*@*.? *") [9] to ensure the correct format. Additionally, I prevent duplicate entries by setting the email field as 'Indexed (No Duplicates), which enforces uniqueness across records.



CUSTOMER

Field Name	Data Type	Description (Optional)
CustomerID	AutoNumber	A unique identifier for each customer
FirstName	Short Text	The customer's first name
LastName	Short Text	The customer's last name
Email	Short Text	A unique multivalued attribute that can store different email addresses per customer
Phone	Short Text	A unique multivalued attribute that can store different phone numbers for each customer
DateOfBirth	Date/Time	The customer date of birth
Gender	Short Text	The gender of the customer
LoyaltyPoints	Number	The number of loyalty points accumulated by the customer

General | Lookup

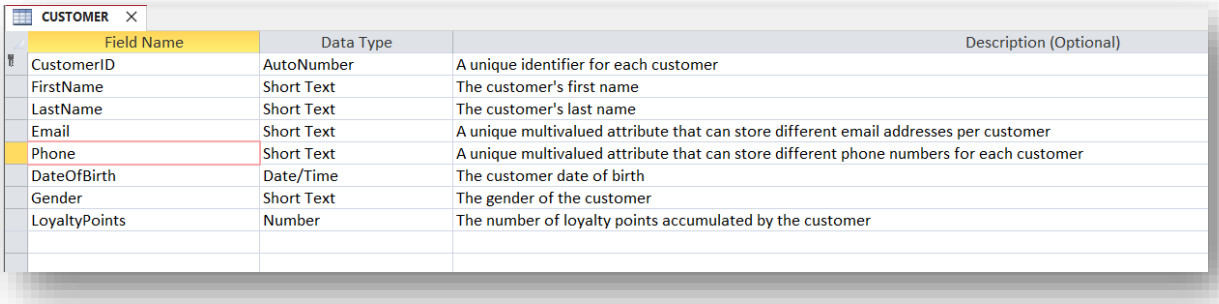
Field Size	255
Format	
Input Mask	
Caption	
Default Value	
Validation Rule	Like "*?@?*.?*"
Validation Text	Please enter a valid email address
Required	Yes
Allow Zero Length	No
Indexed	Yes (No Duplicates)
Unicode Compression	Yes
IME Mode	No Control
IME Sentence Mode	None
Text Align	General

Relationships | CUSTOMER

CustomerID	FristName	LastName	Email	Phone	DateofBirth	Gender	LoyaltyPoin	Cli
1	Alice	Smith	alice.smith@example.com	+44 2079460985	4/12/1985	Female	1500	
2	Bob	Johnson	bob.johnson@example.com	+44 2071234567	8/22/1990	Male	820	
3	Charlie	Wiliams	c.williams@example.com	+44 7800123456	12/5/1975	Male	2300	
4	Diana	Evans	dianaevans@example.com	+44 2074567890	11/30/1988	Female	1250	
5	Ethan	Brown	e.brown!example.com	+44 7900654321	7/19/1995	Male	940	
(New)							0	

3.3.2. VALIDATION OF PHONE NUMBER WITH AN IMPUT MASK

In Microsoft Access, I validated phone numbers using a validation rule (Like "[0-9] *") to ensure only numeric values are entered. I also applied an input mask (+9990000000000;_) to enforce a structured format.[10] Additionally, I set the field as 'Indexed (No Duplicates)' to prevent duplicate phone numbers from being stored.

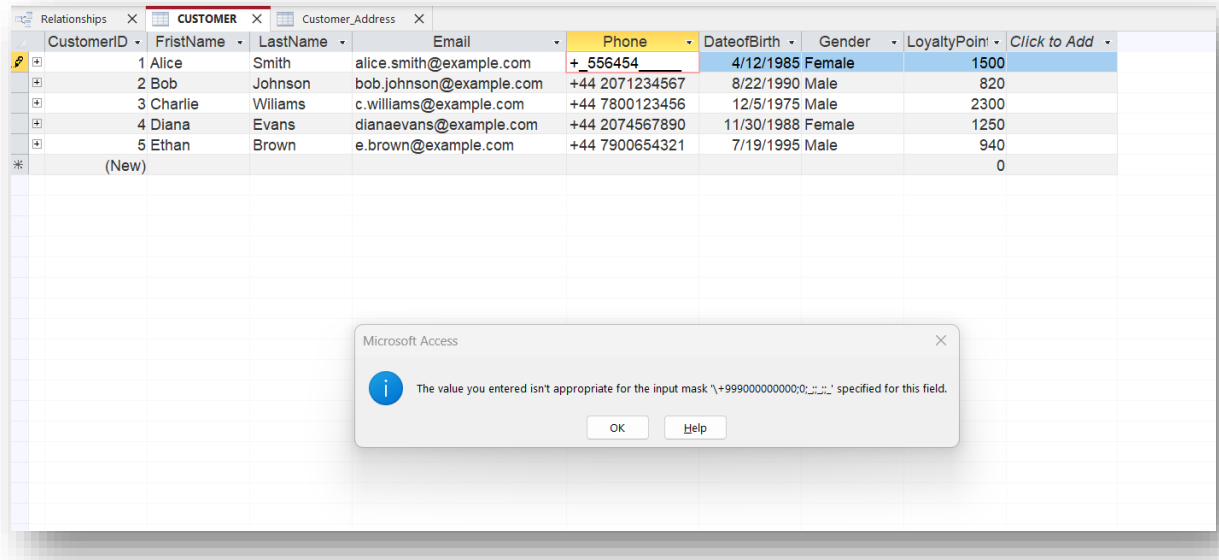


CUSTOMER

Field Name	Data Type	Description (Optional)
CustomerID	AutoNumber	A unique identifier for each customer
FirstName	Short Text	The customer's first name
LastName	Short Text	The customer's last name
Email	Short Text	A unique multivalued attribute that can store different email addresses per customer
Phone	Short Text	A unique multivalued attribute that can store different phone numbers for each customer
DateOfBirth	Date/Time	The customer date of birth
Gender	Short Text	The gender of the customer
LoyaltyPoints	Number	The number of loyalty points accumulated by the customer

General | Lookup

Field Size	255
Format	
Input Mask	\+999000000000;0;_;_;_
Caption	
Default Value	
Validation Rule	Like "+[0-9]*"
Validation Text	Please enter a valid Phone Number, (e.g., +447593640951
Required	Yes
Allow Zero Length	No
Indexed	Yes (Duplicates OK)
Unicode Compression	Yes
IME Mode	No Control
IME Sentence Mode	None
Text Align	General



Relationships | CUSTOMER | Customer_Address

CustomerID	FristName	LastName	Email	Phone	DateofBirth	Gender	LoyaltyPoint	Click to Add
1	Alice	Smith	alice.smith@example.com	+_556454____	4/12/1985	Female	1500	
2	Bob	Johnson	bob.johnson@example.com	+44 2071234567	8/22/1990	Male	820	
3	Charlie	Wiliams	c.williams@example.com	+44 7800123456	12/5/1975	Male	2300	
4	Diana	Evans	dianaevans@example.com	+44 2074567890	11/30/1988	Female	1250	
5	Ethan	Brown	e.brown@example.com	+44 7900654321	7/19/1995	Male	940	
(New)							0	

Microsoft Access

The value you entered isn't appropriate for the input mask '\+999000000000;0;_;_;_' specified for this field.

OK | Help

3.3.3. VALIDATION OF FLAT NUMBER WITH A VALIDATION RULE

I validated flat numbers using a validation rule (Like "[0-9A-Za-z/]*") to allow numbers, letters, and slashes for flexibility

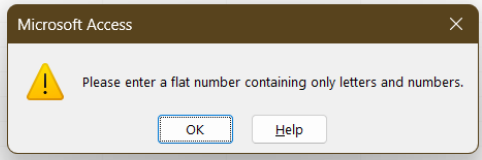
Customer_Address

Field Name	Data Type	Description (Optional)
CustomerAddressID	AutoNumber	A unique identifier for each customer_Address
CustomerID	Number	Foreign Key to link Customer and Customer_Address
FlatNumber	Short Text	Customer flat number
Street	Short Text	Street where customer lives
City	Short Text	City where customert lives
ZipCode	Short Text	Customer zip code

General | Lookup

Field Size	255
Format	
Input Mask	
Caption	
Default Value	
Validation Rule	Like "[0-9A-Za-z/]*"
Validation Text	Please enter Flat Number containing only letters and numbers
Required	Yes
Allow Zero Length	No
Indexed	No
Unicode Compression	Yes
IME Mode	No Control
IME Sentence Mode	None
Text Align	General

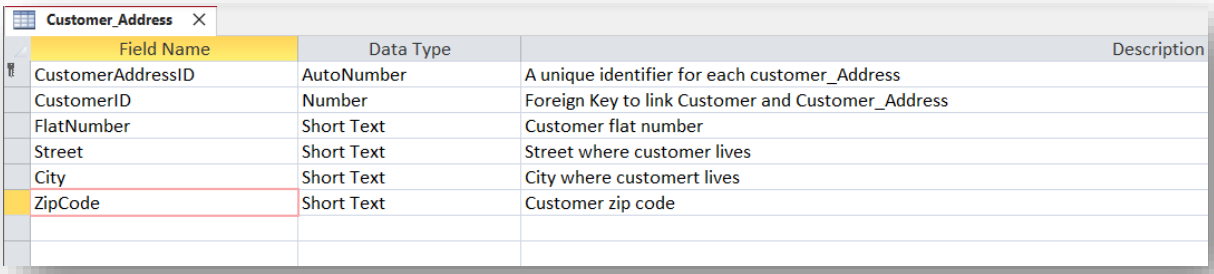
CustomerAc	CustomerID	FlatNumber	Street	City	PostalCode	Click to Add
1	1	!A	Baker Street	London	NW1 6XE	
2	2	5B	Fleet Street	London	EC4 1AA	
3	3	2B	King's Road	Chelsea	SW3 5XP	
4	4	C	Oxford Street	London	W1D 1BS	
5	5	1D	High Street	Kensington	NW8 5TT	
(New)	0					



3.3.4. VALIDATION OF ZIP CODE WITH AN IMPUT MASK

I validated ZIP codes using a validation rule (Like "[0-9A-Za-z]*") to allow numbers, letters, and spaces. To enforce the correct format, I applied an input mask (>LL0 \ 0LL;0;_) for UK postcodes. Since multiple people can live in the same area, I did not set the field as 'Indexed (No Duplicates).

UK ZIP codes (postcodes) have complex formats that vary by region, so I used one as a reference to enforce a basic structure. However, this should be reviewed and adjusted to accommodate all postcode variations.

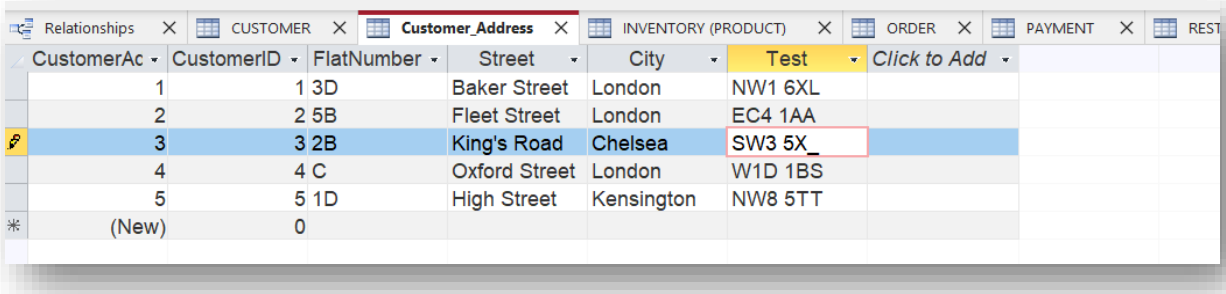


Customer_Address

Field Name	Data Type	Description
CustomerAddressID	AutoNumber	A unique identifier for each customer_Address
CustomerID	Number	Foreign Key to link Customer and Customer_Address
FlatNumber	Short Text	Customer flat number
Street	Short Text	Street where customer lives
City	Short Text	City where customert lives
ZipCode	Short Text	Customer zip code

General | Lookup

Field Size	255
Format	
Input Mask	LL0" "0LL;0;_
Caption	
Default Value	
Validation Rule	Like "[0-9A-Za-z]*"
Validation Text	
Required	Yes
Allow Zero Length	No
Indexed	Yes (Duplicates OK)
Unicode Compression	Yes
IME Mode	No Control
IME Sentence Mode	None
Text Align	General



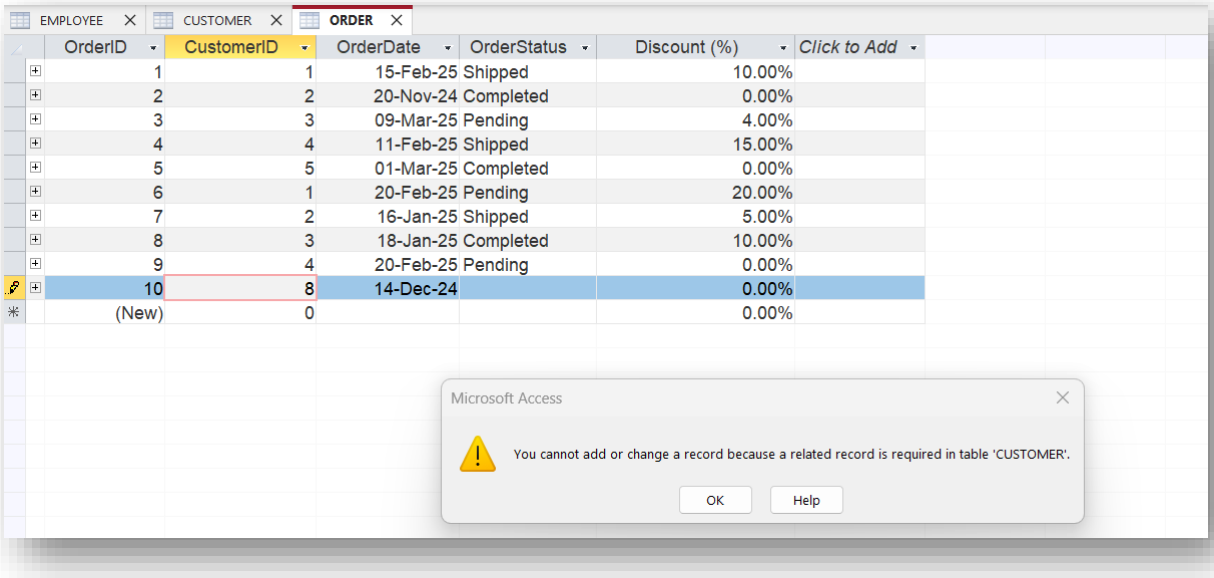
Relationships | CUSTOMER | Customer_Address | INVENTORY (PRODUCT) | ORDER | PAYMENT | REST

CustomerAc	CustomerID	FlatNumber	Street	City	Test	Click to Add
1	1	3D	Baker Street	London	NW1 6XL	
2	2	5B	Fleet Street	London	EC4 1AA	
3	3	2B	King's Road	Chelsea	SW3 5X_	
4	4	C	Oxford Street	London	W1D 1BS	
5	5	1D	High Street	Kensington	NW8 5TT	
(New)	0					

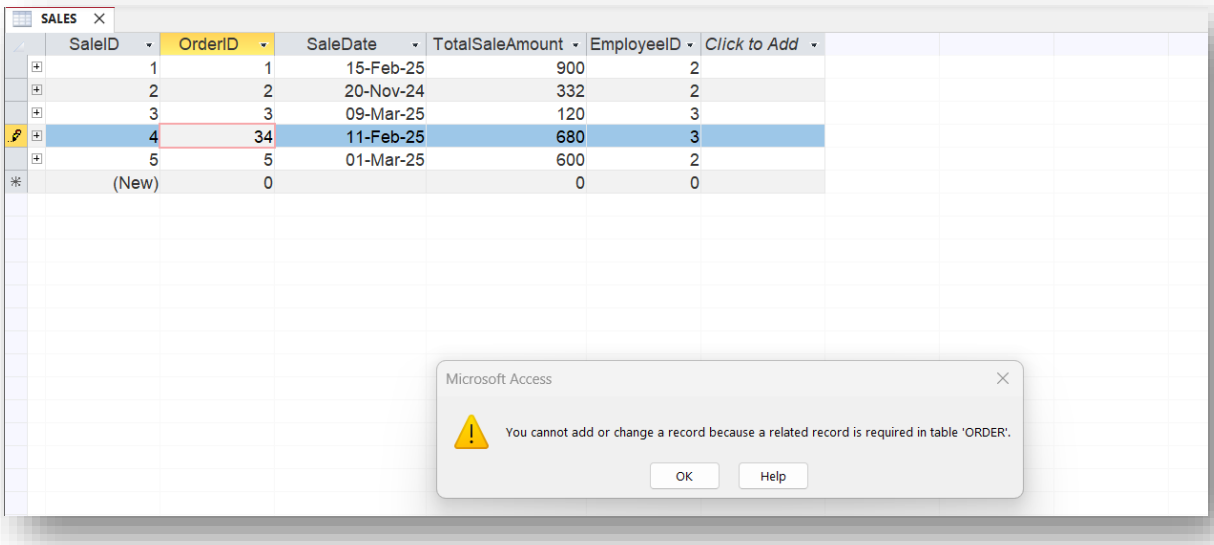
3.4. CHECKING RELATIONSHIPS

To verify that referential integrity [11]is enforced correctly in the database, I tested the relationships by attempting to change a foreign key value to one that does not exist in the related table. This helped confirm that the system correctly prevents invalid references, ensuring data consistency.

3.4.1. CUSTOMER – ORDER



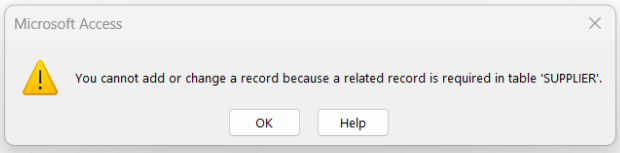
3.4.2. ORDER – SALES



3.4.3. SUPPLIER – INVENTORY

INVENTORY (PRODUCT)

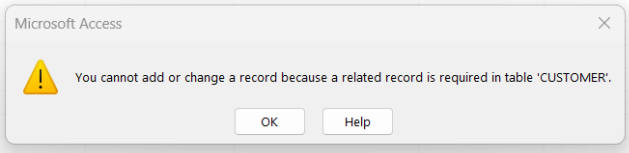
ProductID	ProductName	Category	SupplierID	Price	Cost	StockQuantity	Pair
1	Diamond Stud Earrings	Earrings	2	$500.00	$300.00	15	
2	Gold Pendant Necklace	Necklaces	3	$350.00	$200.00	20	
3	Silver Charm Bracelet	Bracelets	4	$120.00	$70.00	25	
4	Platinum Wedding Band	Rings	5	$800.00	$500.00	10	
5	Pearl Drop Earrings	Earrings	6	$200.00	$150.00	12	
6	Sapphire Cocktail Ring	Rings	2	$600.00	$120.00	18	
7	Ruby Tennis Bracelet	Bracelets	3	$750.00	$450.00	8	
8	Emerald Pendant	Necklaces	4	$400.00	$250.00	14	
9	Opal Stud Earrings	Earrings	25	$180.00	$100.00	22	
10	Amethyst Anklet	Anklets	2	$130.00	$80.00	16	
(New)			0	$0.00	$0.00	0	



3.4.4. CUSTOMER – PAYMENT

ORDER | PAYMENT

PaymentID	OrderID	PaymentDate	CustomerID	PaymentStatus	Click to Add
1	1	15-Feb-25	1	Completed	
2	2	16-Feb-25	2	Completed	
3	3	17-Feb-25	3	Pending	
4	4	18-Feb-25	4	Completed	
5	5	19-Feb-25	9	Completed	
6	6	20-Feb-25	1	Pending	
7	7	21-Feb-25	2	Completed	
8	8	22-Feb-25	3	Completed	
9	9	21-Feb-25	4	Pending	
10	10	23-Feb-25	5	Completed	
(New)	0		0		



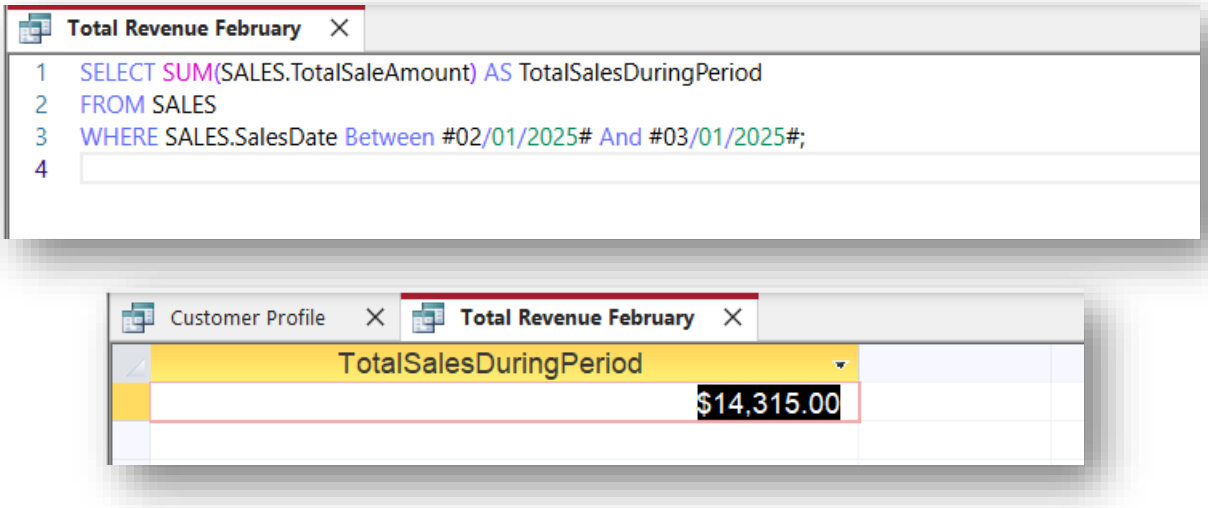
4. MICROSOFT ACCESS FUNCTIONALITY

4.1. QUERIES

The selection of these queries is based on the most relevant and used Key Performance Indicators (KPIs) [12] In business analysis. These queries provide a comprehensive view of the company's operational and financial performance by focusing on sales performance, economic health, employee productivity, customer demographics, and product profitability. Each KPI-driven query enables data-driven decision-making, ensuring strategic improvements and business growth.

4.1.1. CURRENT MONTHLY SALES (from February 1st to March 1st)

Tracking monthly sales is important to see how the business is growing and understand revenue patterns. It helps spot trends and adjust marketing plans accordingly. Regularly checking sales ensures steady cash flow and financial stability. Looking at monthly sales also helps set realistic revenue goals. By analyzing sales, businesses can make better decisions and plan effectively.



This query calculates the total sales revenue generated within a specific date range. It sums up all sales transactions between January 2 and January 3, 2025.

What This Query Tells Us:

- The total amount of money earned from sales within the specified period.

How It Works:

- Filters the SALES table to only include records where SalesDate falls between January 2, 2025, and January 3, 2025.
- Sums the TotalSaleAmount of the filtered transactions to calculate total revenue during this timeframe.

What We Get:

A single value representing the total sales revenue for the given period.

4.1.2. EMPLOYEE PERFORMANCE

Rewarding employees for their sales encourages them to work harder and be more productive. This analysis shows which employees are performing the best and calculates their 5% bonus based on their total sales. Recognizing top performers creates a competitive and motivating environment, encouraging everyone to contribute more to the company's success. This information can also be used for performance reviews, promotions, and figuring out where employees might need extra training.

Employee Performance

```
1  SELECT SALES.EmployeeID, EMPLOYEE.FirstName, EMPLOYEE.LastName, Format(SALES.SalesDate,'yyyy-mm') AS SalesMonth,
2  Sum(SALES.TotalSaleAmount) AS TotalSales, Sum(SALES.TotalSaleAmount)*0.05 AS Bonus, (EMPLOYEE.Salary/12)+(Sum(SALES.TotalSaleAmount)*0.05) AS MonthlySalaryWithBonus
3  FROM EMPLOYEE INNER JOIN SALES ON EMPLOYEE.EmployeeID = SALES.EmployeeID
4  WHERE (((SALES.SalesDate) Between #1/1/2025# And #2/28/2025#))
5  GROUP BY SALES.EmployeeID, EMPLOYEE.FirstName, EMPLOYEE.LastName, Format(SALES.SalesDate,'yyyy-mm'), EMPLOYEE.Salary
6  ORDER BY Sum(SALES.TotalSaleAmount) DESC;
7
```

Employee Performance

EmployeeID	FirstName	LastName	SalesMonth	TotalSales	Bonus	MonthlySalaryWithBonus
1	Diana	Johnson	2025-01	$13,322.40	666.12	4416.12
2	Bob	Williamsburg	2025-01	$8,955.90	447.795	3947.795
3	Charlie	Brown	2025-01	$6,922.20	346.11	3512.77666666667
2	Bob	Williamsburg	2025-02	$5,235.00	261.75	3761.75
3	Charlie	Brown	2025-02	$4,978.00	248.9	3415.56666666667
1	Diana	Johnson	2025-02	$4,102.00	205.1	3955.1

This query calculates monthly sales performance and bonuses for employees within a specified period (January 1, 2025, to February 28, 2025). It provides details on individual employees' total sales, earned bonuses, and updated monthly salary including the bonus.

What This Query Tells Us:

- EmployeeID – Unique identifier of the employee.
- FirstName & LastName – Employee's first and last name.
- SalesMonth – The month in which sales were made, formatted as yyyy-mm.
- TotalSales – The total sales amount generated by each employee during that month.
- Bonus – The employee's bonus, calculated as 5% of their total sales.
- MonthlySalaryWithBonus – The employee's total monthly earnings, which includes their base salary (divided by 12 for monthly salary) plus the bonus.

How It Works:

Joins the EMPLOYEE and SALES tables to link employees with their sales records.

- Filters sales records to only include transactions made between January 1, 2025, and February 28, 2025.
- Groups data by EmployeeID, FirstName, LastName, SalesMonth, and Salary to calculate performance per employee per month.
- Calculates:
    - TotalSales: Sum of TotalSaleAmount per employee per month.
    - Bonus: 5% of total sales for each employee.
    - MonthlySalaryWithBonus: Base salary divided by 12 months plus the bonus.

- Sorts the results by TotalSales in descending order, showing the highest-performing employees first.

What We Get:

- A table listing employees along with their monthly sales, earned bonuses, and updated salary including their performance-based bonus.

4.1.3. SUPPLIER PERFORMANCE (Where We Spend the Most Money)

Managing supplier relationships is key to keeping costs down and improving how a business operates. This analysis helps identify which supplier a business spends the most money with, allowing the company to:

- Negotiate better deals or discounts for buying in bulk.
- Understand how much they rely on certain suppliers and the risks involved.
- Improve buying strategies to reduce costs.

Popular Supplier

```
1  SELECT SUPPLIER.SupplierID, SUPPLIER.FirstName, SUPPLIER.LastName, SUM(INVENTORY.Cost * INVENTORY.StockQuantity) AS TotalSpent,
2  SUM(INVENTORY.Price * INVENTORY.StockQuantity) AS TotalRevenue, (SUM(INVENTORY.Price * INVENTORY.StockQuantity) - SUM(INVENTORY.Cost * INVENTORY.StockQuantity)) AS Profit
3  FROM SUPPLIER INNER JOIN INVENTORY ON SUPPLIER.SupplierID = INVENTORY.SupplierID
4  GROUP BY SUPPLIER.SupplierID, SUPPLIER.FirstName, SUPPLIER.LastName
5  ORDER BY SUM(INVENTORY.Cost * INVENTORY.StockQuantity) DESC;
6
```

Supplier Performance

SupplierID	FirstName	LastName	TotalSpent	TotalRevenue	Profit
3	Sophia	Cartier	$31,200.00	$47,520.00	$16,320.00
2	Ethan	Goldstein	$22,500.00	$37,360.00	$14,860.00
1	Olivia	Sterling	$18,800.00	$31,260.00	$12,460.00
5	Ava	Marquise	$16,790.00	$27,450.00	$10,660.00
4	Liam	Argent	$14,660.00	$24,080.00	$9,420.00

This query analyzes supplier-related inventory costs, revenues, and profits, showing how much was spent on inventory, its potential revenue, and the resulting profit for each supplier.

What This Query Tells Us:

- SupplierID – Unique identifier for each supplier.
- FirstName & LastName – Supplier's first and last name.
- TotalSpent – The total amount spent on inventory from this supplier, calculated as: ∑(Cost×StockQuantity
- TotalRevenue – The potential revenue from selling the inventory, calculated as: ∑(Price×StockQuantity
- Profit – The estimated profit from the supplier's inventory, calculated as: TotalRevenue−TotalSpent

How It Works:

- Joins the SUPPLIER and INVENTORY tables to connect each supplier with their supplied inventory.
- Groups data by SupplierID, FirstName, and LastName to calculate financial metrics for each supplier.
- Calculates:
    - TotalSpent: Sum of cost price × stock quantity for each supplier.
    - TotalRevenue: Sum of selling price × stock quantity for each supplier.
    - Profit: The difference between total revenue and total spent.
- Sorts the results by TotalSpent in descending order, listing suppliers with the highest inventory cost at the top.

What We Get:

A table displaying each supplier's financial impact, showing how much, they've contributed to stock costs, potential sales revenue, and the estimated profit from their inventory.

4.1.4. PRODUCT PERFORMANCE

Analyzing how well products are selling is important for managing stock and maximizing profits. This analysis helps businesses:

- Best-selling product: By focusing on products that sell the most, businesses can keep enough stock to prevent running out and promote those items more.
- Most profitable product: Knowing which products bring in the most profit helps businesses focus on selling more of those high-margin items.
- Least profitable product: Identifying products that aren't selling well helps businesses decide whether to stop selling them or make changes to avoid losing money.

Product Performance

```
1  SELECT Order_Items.ProductID, INVENTORY.ProductName, SUM(Order_Items.Quantity) AS TotalQuantitySold,
2  SUM(Order_Items.Quantity * Order_Items.Price) AS TotalRevenue,
3  SUM(Order_Items.Quantity * (INVENTORY.Price - INVENTORY.Cost)) AS TotalProfit,
4  (SUM(Order_Items.Quantity * (INVENTORY.Price - INVENTORY.Cost)) / SUM(Order_Items.Quantity * Order_Items.Price)) * 100 AS ProfitPercentage
5  FROM ([ORDER] INNER JOIN Order_Items ON ORDER.OrderID = Order_Items.OrderID) INNER JOIN INVENTORY ON INVENTORY.ProductID = Order_Items.ProductID
6  GROUP BY Order_Items.ProductID, INVENTORY.ProductName
7  ORDER BY SUM(Order_Items.Quantity * Order_Items.Price) DESC;
8
```

Product Performance

ProductID	ProductName	TotalQuantitySold	TotalRevenue	TotalProfit	ProfitPercentage
6	Platinium Wedding Band	6	$9,000.00	$3,000.00	33.3333333333333
1	Diamond Ring	5	$6,000.00	$2,000.00	33.3333333333333
7	Ruby Brooch	5	$3,900.00	$1,650.00	42.3076923076923
2	Gold Necklace	4	$3,800.00	$1,400.00	36.8421052631579
11	Topaz Pendant	4	$2,880.00	$1,080.00	37.5
9	Emerald Anklet	5	$2,800.00	$1,050.00	37.5
13	Tanzanite Earrings	4	$2,560.00	$960.00	37.5
5	Sapphire Pendants	3	$2,550.00	$1,050.00	41.1764705882353
15	Citrine Ring	4	$2,320.00	$920.00	39.6551724137931

What This Query Tells Us:

- ProductID – Unique identifier for each product.
- ProductName – The name of the product.
- TotalQuantitySold – The total number of units sold for each product.
- TotalRevenue – The total revenue generated from sales of the product.
- TotalProfit – The total profit made from selling the product, calculated as (Selling Price - Cost Price) × Quantity Sold.
- ProfitPercentage – The profit margin for each product, calculated as (TotalProfit / TotalRevenue) × 100.

How It Works:

- Joins the ORDER, Order_Items, and INVENTORY tables to link sales transactions with product details.
- Groups data by ProductID and ProductName to calculate sales performance for each product.
- Calculate key metrics:
    - TotalQuantitySold: Sum of Quantity sold per product.
    - TotalRevenue: Sum of Quantity × Selling Price for each product.
    - TotalProfit: Sum of (Selling Price - Cost Price) × Quantity Sold for each product.
    - ProfitPercentage: Ratio of profit to revenue, expressed as a percentage.
- Sorts results by TotalRevenue in descending order to highlight the highest-earning products.

What We Get:

A table displaying each product's total sales, revenue, profit, and profit percentage, helping analyse product profitability and sales performance.

4.1.5. CUSTOMER PROFILE

Understanding "who your customers are" helps businesses create better marketing strategies, offer personalized experiences, and improve product selection. This analysis groups customers by gender and age and looks at how much they spend overall and on average. This helps businesses target the right customers with the right products and offers.

Customer Profile

```
1  SELECT CUSTOMER.Gender, Sum(SALES.TotalSaleAmount) AS TotalSpent, DateDiff('yyyy',CUSTOMER.DateOfBirth,Date()) AS Age, Order_Items.Category
2  FROM ((CUSTOMER INNER JOIN [ORDER] ON CUSTOMER.CustomerID = ORDER.CustomerID)
3  INNER JOIN (INVENTORY INNER JOIN Order_Items ON INVENTORY.ProductID = Order_Items.ProductID) ON ORDER.OrderID = Order_Items.OrderID)
4  INNER JOIN SALES ON ORDER.OrderID = SALES.OrderID
5  GROUP BY CUSTOMER.CustomerID, CUSTOMER.DateOfBirth, CUSTOMER.Gender, Order_Items.ProductID, INVENTORY.ProductName,
6  DateDiff('yyyy',CUSTOMER.DateOfBirth,Date()), Order_Items.Category
7  ORDER BY Sum(SALES.TotalSaleAmount) DESC;
8
```

Customer Profile

Gender	TotalSpent	Age	Category
Male	$2,303.50	35	Pendants
Female	$2,232.00	43	Rings
Female	$2,112.00	42	Charms
Female	$1,977.00	36	Accessories
Male	$1,940.00	33	Necklaces
Female	$1,862.00	32	Pendants
Female	$1,836.80	34	Anklets
Male	$1,800.00	35	Rings
Female	$1,568.00	40	Earrings

This query retrieves data on customer spending, age, and product categories. It calculates each customer's total amount spent, their age, and the category of products they purchased

What This Query Tells Us:

- Gender – The gender of each customer.
- TotalSpent – The total amount a customer has spent on purchases.
- Age – The customer's age, calculated by subtracting their birth year from the current year.
- Category – The type of product they bought.

How It Works:

- Joins multiple tables (CUSTOMER, ORDER, Order_Items, INVENTORY, SALES) to connect customers with their purchases.
- Groups the data by customer, ensuring each row represents one customer's spending on a specific product category.
- Calculates:
    - Total amount spent by summing SALES.TotalSaleAmount for each customer.
    - Customer age using DateDiff('yyyy', CUSTOMER.DateOfBirth, Date()).
- Sorts the results in descending order of TotalSpent, so the highest spenders appear first.

What We Get:

The output is a table showing each customer's gender, age, total spending, and product category. The biggest spenders are listed at the top.

4.1.6. CUSTOMER AVERAGE

This query looks at customer information and spending habits from sales data. It calculates the average age, how much customers spend on average, and how many customers there are. Then, it groups the data by gender and product category and lists the results starting with the highest spenders.

Average Customer Profile

```
1  SELECT Int(AVG(DateDiff('yyyy', CUSTOMER.DateOfBirth, Date()))) AS AverageAge,
2  AVG(SALES.TotalSaleAmount) AS AverageExpenditure, CUSTOMER.Gender, Order_Items.Category,
3  COUNT(CUSTOMER.CustomerID) AS NumberOfCustomers
4  FROM (((CUSTOMER INNER JOIN [ORDER] ON CUSTOMER.CustomerID = [ORDER].CustomerID)
5  INNER JOIN Order_Items ON [ORDER].OrderID = Order_Items.OrderID)
6  INNER JOIN INVENTORY ON Order_Items.ProductID = INVENTORY.ProductID)
7  INNER JOIN SALES ON [ORDER].OrderID = SALES.OrderID
8  GROUP BY CUSTOMER.Gender, Order_Items.Category
9  ORDER BY AVG(SALES.TotalSaleAmount) DESC;
10
```

Average Customer Profile

AverageAge	AverageExpenditure	Gender	Category	NumberOfCustomers
31	$1,544.40	Male	Broches	1
37	$1,507.00	Female	Pendants	2
38	$1,408.40	Female	Rings	5
31	$1,399.20	Male	Rings	5
40	$1,340.33	Female	Anklets	3
33	$1,269.50	Male	Earrings	4
28	$1,218.50	Male	Anklets	2
32	$1,158.00	Male	Bracelets	3
36	$1,155.25	Female	Earrings	2
42	$1,110.00	Female	Brooches	2
31	$1,063.33	Male	Necklaces	3

What This Query Tells Us:

- AverageAge – The average age of customers, rounded to a whole number.
- AverageExpenditure – The typical amount a customer spends.
- Gender – Whether the customer is male or female.
- Category – The type of product they bought.
- NumberOfCustomers – How many customers fall into each gender-category group.

How It Works:

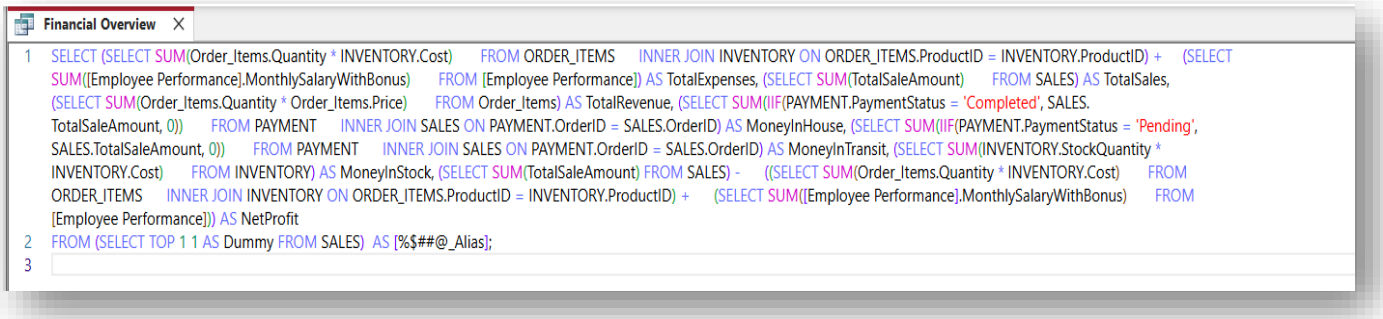
- It joins different tables (CUSTOMER, ORDER, Order_Items, INVENTORY, SALES) to match customers' purchases.
- It groups the data by gender and product category.
- It calculates:
  - The average age using AVG(DateDiff('yyyy', CUSTOMER.DateOfBirth, Date())).
  - The average amount spent using AVG(SALES.TotalSaleAmount).
  - The number of customers using COUNT(CUSTOMER.CustomerID).
- Orders the results from the highest to the lowest spending groups.

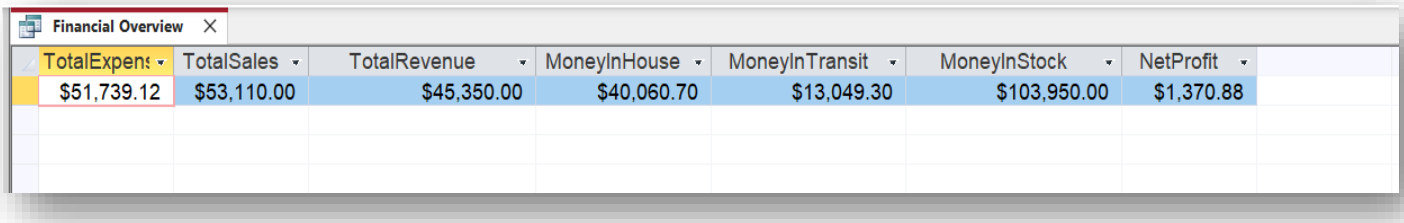
What We Get:

The output is a table that shows gender, product category, average age, average spending per sale, and the number of customers in each group. The biggest spenders appear first.

4.1.7. FINANCIAL OVERWIEW

This query provides a detailed breakdown of the financial status of the Company.





What This Query Tells Us:

- TotalExpenses: The total amount the company spends, including inventory costs and employee salaries (with bonuses).
- TotalSales: The total amount of sales made by the company.
- TotalRevenue: The total revenue from all the products sold (calculated as the quantity of items sold times their price).
- MoneyInHouse: The total amount of money the company has already received from completed payments.
- MoneyInTransit: The total amount of money that's pending, e.g., payments that are still in process.
- MoneyInStock: The total value of the company's inventory, calculated by multiplying the stock quantity of each product by its cost.
- NetProfit: The total profit the company made, calculated as the difference between the total sales and the sum of expenses.

How It Works:

- Joins multiple subqueries: Each part of the query sums up different pieces of information from various tables like ORDER_ITEMS, INVENTORY, Employee Performance, SALES, and PAYMENT.
- Calculations:
    - TotalExpenses: Adds the cost of inventory and the monthly salaries of employees with bonuses.
    - TotalSales: Sums the total sales amount.
    - TotalRevenue: Calculates the revenue from the sales (quantity * price).
    - MoneyInHouse: Sums up the money from completed payments.
    - MoneyInTransit: Sums up the money from payments that are still pending.
    - MoneyInStock: Calculates the total value of inventory on hand.
    - NetProfit: Subtracts the total expenses (inventory + employee salaries) from the total sales to calculate profit.

- No need for complex joins: Instead of connecting multiple tables with standard JOIN operations, this query uses subqueries (the SELECT statements within parentheses) to pull the necessary data and calculate totals.
- Groupings: Each subquery pulls different data, and no grouping is needed because the final result is just a single row showing all these totals.
- Final Calculation: The last part of the query calculates NetProfit by subtracting TotalExpenses from TotalSales.
- Ensure that the relationship between EmployeePerformance.EmployeeID and Sales.EmployeeID is valid.
- Make sure that the cost of the products in INVENTORY.Cost is correctly populated for all products in stock.

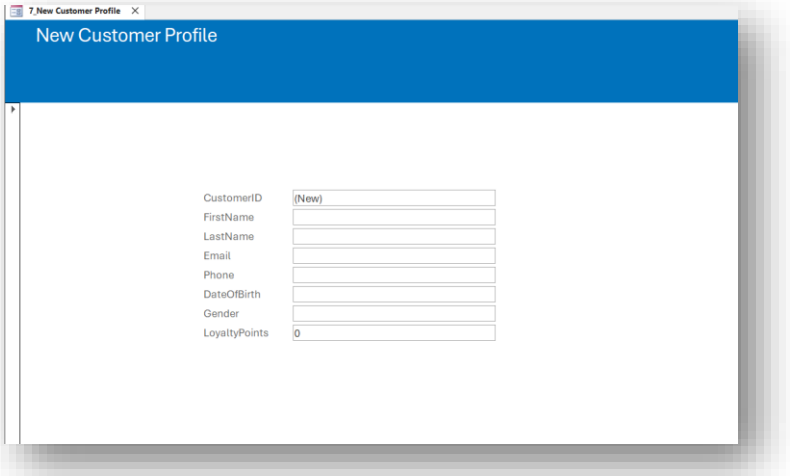
What We Get:

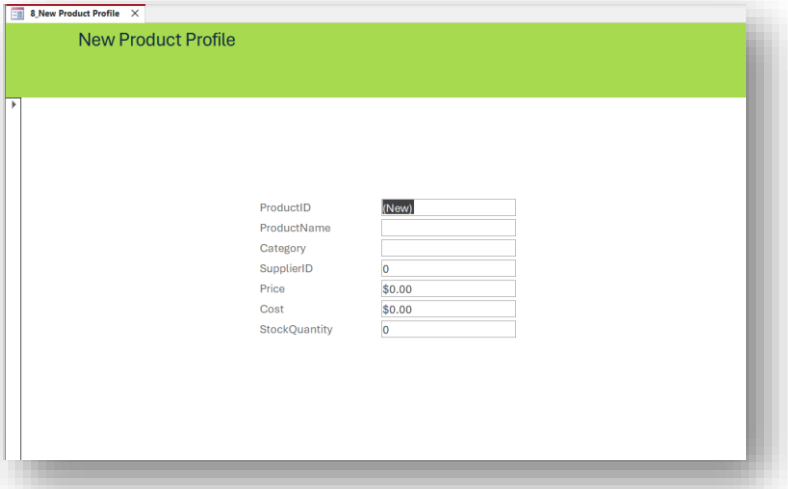
The output is a single row showing the Total Expenses, Sales, Revenue, Money in House, Money in Transit, Money in Stock, and Net Profit. This gives a clear picture of the company's financial situation, including the amount spent, earned, and net profit.

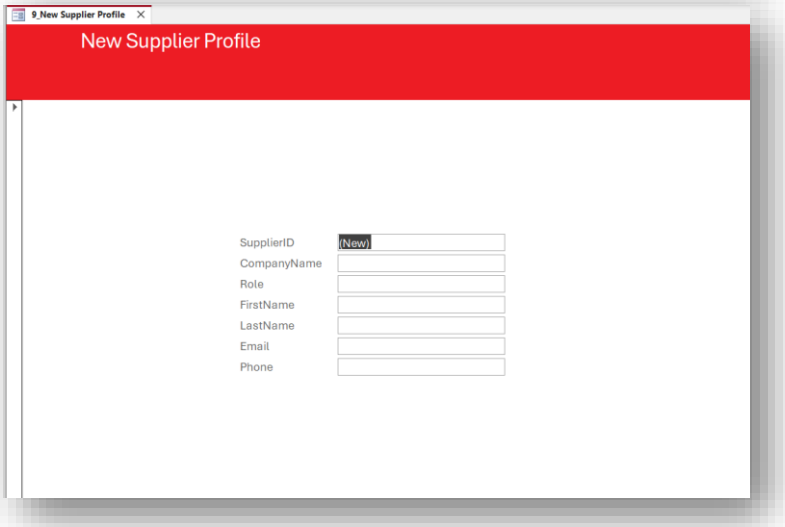
4.2. FORMS AND REPORTS

4.2.1. FORMS

In my project, I used Forms to make entering, viewing, and editing data easier. Forms provide a simple, organised way to interact with the data without dealing directly with the tables. They help reduce errors and make the process more efficient.[13]







With these forms, we can:

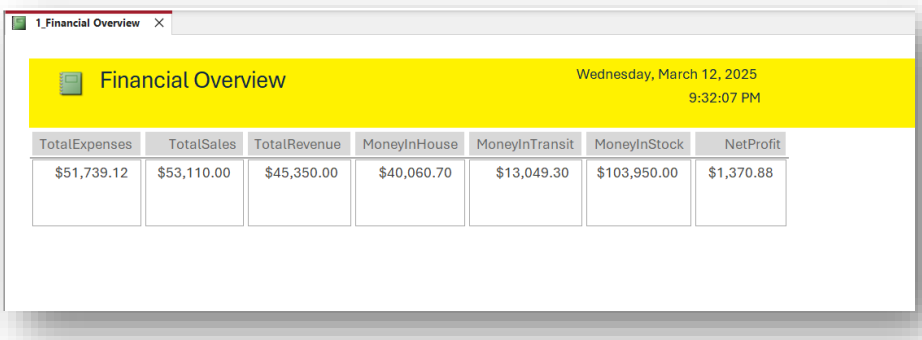
- Create New Customers: We can easily add new customer details to the database, and the form will automatically create a unique Customer ID for each customer.
- Create New Products: We can quickly add new products to the product list without manually updating the table, and an automatic Product ID is created for each product.
- Create New Suppliers: We can add new suppliers to the system, and the form will make sure a unique Supplier ID is created for each supplier.

Using forms allows us to update the database directly while also following referential integrity rules. This means the data in different tables stays connected and correct (for example, ensuring a customer ID in an order matches an existing customer in the customer table). The automatic creation of IDs ensures that every record is unique, which helps avoid mistakes and keeps the database organized.

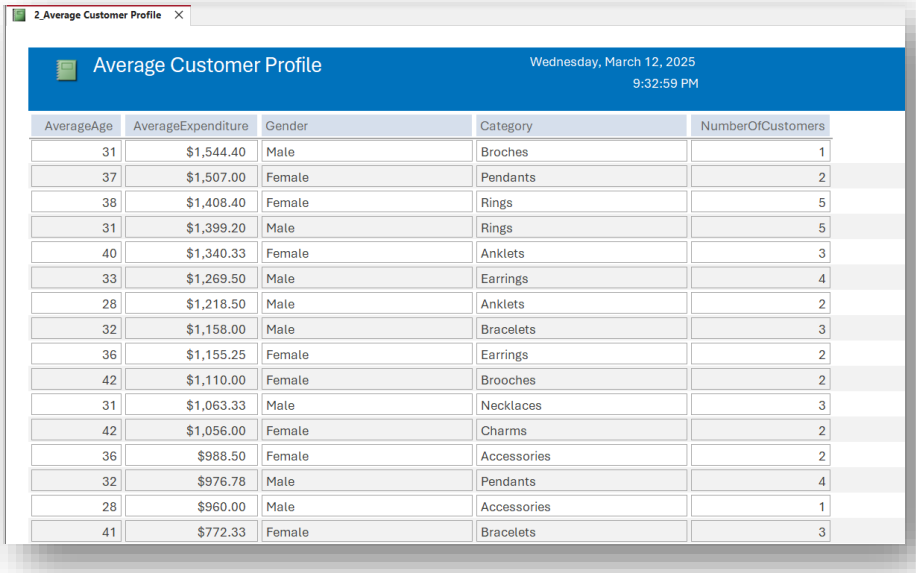
These features make entering and managing data quicker, more accurate, and less likely to have errors.

4.2.2. REPORTS

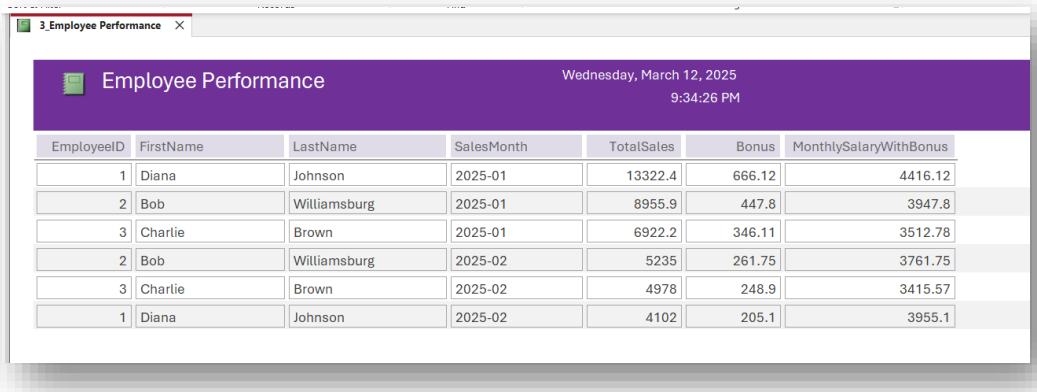
I also created Reports to help me organize and present the data in a clean, readable format. Reports summarize data and make it easy to share and print.[14]. All the reports are based on the queries before mentioned.



TotalExpenses	TotalSales	TotalRevenue	MoneyInHouse	MoneyInTransit	MoneyInStock	NetProfit
$51,739.12	$53,110.00	$45,350.00	$40,060.70	$13,049.30	$103,950.00	$1,370.88



AverageAge	AverageExpenditure	Gender	Category	NumberOfCustomers
31	$1,544.40	Male	Broches	1
37	$1,507.00	Female	Pendants	2
38	$1,408.40	Female	Rings	5
31	$1,399.20	Male	Rings	5
40	$1,340.33	Female	Anklets	3
33	$1,269.50	Male	Earrings	4
28	$1,218.50	Male	Anklets	2
32	$1,158.00	Male	Bracelets	3
36	$1,155.25	Female	Earrings	2
42	$1,110.00	Female	Brooches	2
31	$1,063.33	Male	Necklaces	3
42	$1,056.00	Female	Charms	2
36	$988.50	Female	Accessories	2
32	$976.78	Male	Pendants	4
28	$960.00	Male	Accessories	1
41	$772.33	Female	Bracelets	3

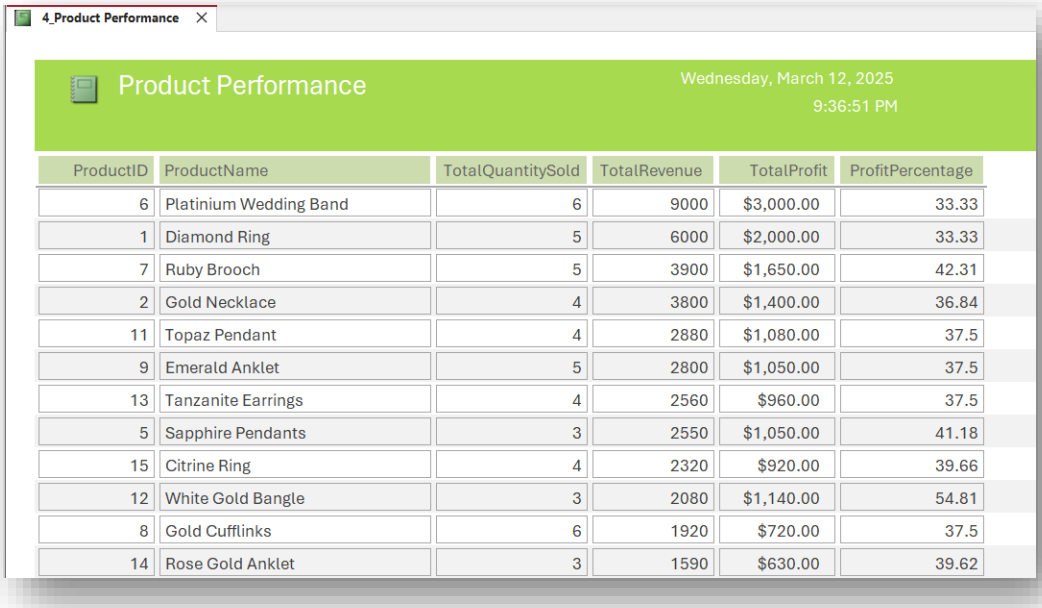


3_Employee Performance

Employee Performance

Wednesday, March 12, 2025
9:34:26 PM

EmployeeID	FirstName	LastName	SalesMonth	TotalSales	Bonus	MonthlySalaryWithBonus
1	Diana	Johnson	2025-01	13322.4	666.12	4416.12
2	Bob	Williamsburg	2025-01	8955.9	447.8	3947.8
3	Charlie	Brown	2025-01	6922.2	346.11	3512.78
2	Bob	Williamsburg	2025-02	5235	261.75	3761.75
3	Charlie	Brown	2025-02	4978	248.9	3415.57
1	Diana	Johnson	2025-02	4102	205.1	3955.1

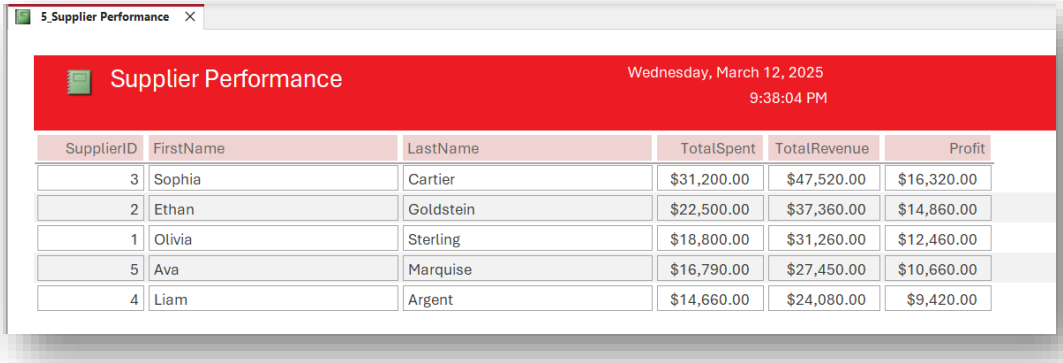


4_Product Performance

Product Performance

Wednesday, March 12, 2025
9:36:51 PM

ProductID	ProductName	TotalQuantitySold	TotalRevenue	TotalProfit	ProfitPercentage
6	Platinium Wedding Band	6	9000	$3,000.00	33.33
1	Diamond Ring	5	6000	$2,000.00	33.33
7	Ruby Brooch	5	3900	$1,650.00	42.31
2	Gold Necklace	4	3800	$1,400.00	36.84
11	Topaz Pendant	4	2880	$1,080.00	37.5
9	Emerald Anklet	5	2800	$1,050.00	37.5
13	Tanzanite Earrings	4	2560	$960.00	37.5
5	Sapphire Pendants	3	2550	$1,050.00	41.18
15	Citrine Ring	4	2320	$920.00	39.66
12	White Gold Bangle	3	2080	$1,140.00	54.81
8	Gold Cufflinks	6	1920	$720.00	37.5
14	Rose Gold Anklet	3	1590	$630.00	39.62



5_Supplier Performance

Supplier Performance

Wednesday, March 12, 2025
9:38:04 PM

SupplierID	FirstName	LastName	TotalSpent	TotalRevenue	Profit
3	Sophia	Cartier	$31,200.00	$47,520.00	$16,320.00
2	Ethan	Goldstein	$22,500.00	$37,360.00	$14,860.00
1	Olivia	Sterling	$18,800.00	$31,260.00	$12,460.00
5	Ava	Marquise	$16,790.00	$27,450.00	$10,660.00
4	Liam	Argent	$14,660.00	$24,080.00	$9,420.00

5. CONCLUSION

For this project, I created a Microsoft Access database to manage Craft Gem's key business data like Customers, Products, Suppliers, Orders, and Sales. The data is organized into separate, well-structured tables that are logically linked to one another, making the system easier to navigate and work with.

To improve efficiency, I applied normalization, which helped reduce data redundancy. This means I created separate tables for key business subjects, like Customers and Products, and then linked them using unique IDs. This way, the database stays clean without repeating data and works more smoothly.

I used referential integrity to connect the tables, which ensures the data stays consistent and avoids errors. For example, each order is correctly linked to an existing customer, and each order item refers to a specific product. This setup helps prevent mistakes and ensures everything runs properly.

To make data entry more user-friendly, I created Forms that let users add, update, and delete data without having to interact directly with the tables. The forms automatically generate unique IDs for new customers, products, or suppliers, reducing the risk of errors and making everything more streamlined.

Additionally, I built Reports to summarize key business information, such as financial overview, customer profiles, employee performance, product performance, and supplier performance. These reports help provide a clear view of the data, making it easier to share insights and helping make better, data-driven decisions.

I also conducted validation and testing during the project to ensure everything was working as expected. I verified that the data was correct, the forms and tables were functioning correctly, and all the calculations were accurate.

In conclusion, a well-organized database, built with normalization, referential integrity, and thorough testing, is essential for managing data efficiently and accurately. With the use of Forms and Reports, the database is easy to use and helps businesses make more intelligent decisions by presenting important information in an organized way. Overall, this system will help Craft Gem track performance trends and customer insights, making it easier to meet their needs and make informed decisions.

6. REFERENCES

[1] "Database basics - Microsoft Support." Accessed: Mar. 06, 2025. [Online]. Available: https://support.microsoft.com/en-gb/office/database-basics-a849ac16-07c7-4a31-9948-3c8c94a7c204

[2] "Informix Servers 14.10." Accessed: Mar. 06, 2025. [Online]. Available: https://www.ibm.com/docs/en/informix-servers/14.10?topic=integrity-referential

[3] "What is an Entity Relationship Diagram? | IBM." Accessed: Mar. 03, 2025. [Online]. Available: https://www.ibm.com/think/topics/entity-relationship-diagram

[4] M. Anderson, "Peter P. Chen," IEEE Computer Society. Accessed: Mar. 03, 2025. [Online]. Available: https://www.computer.org/profiles/peter-chen/

[5] Dr. Daniel Soper, *Database Lesson #2 of 8 - The Relational Model*, (May 31, 2013). Accessed: Mar. 07, 2025. [Online Video]. Available: https://www.youtube.com/watch?v=kyGVhx5LwXw

[6] "What is an Entity Relationship Diagram? | IBM." Accessed: Mar. 07, 2025. [Online]. Available: https://www.ibm.com/think/topics/entity-relationship-diagram

[7] "What is Third Normal Form (3NF)? A Beginner-Friendly Guide." Accessed: Mar. 07, 2025. [Online]. Available: https://www.datacamp.com/tutorial/third-normal-form

[8] "z/OS 2.4.0." Accessed: Mar. 07, 2025. [Online]. Available: https://www.ibm.com/docs/en/zos/2.4.0?topic=characteristics-single-valued-multi-valued-attributes

[9] "Restrict data input by using validation rules - Microsoft Support." Accessed: Mar. 08, 2025. [Online]. Available: https://support.microsoft.com/en-gb/office/restrict-data-input-by-using-validation-rules-b91c6b15-bcd3-42c1-90bf-e3a0272e988d

[10] "Control data entry formats with input masks - Microsoft Support." Accessed: Mar. 08, 2025. [Online]. Available: https://support.microsoft.com/en-gb/office/control-data-entry-formats-with-input-masks-e125997a-7791-49e5-8672-4a47832de8da

[11] "Informix Servers 14.10." Accessed: Mar. 11, 2025. [Online]. Available: https://www.ibm.com/docs/en/informix-servers/14.10?topic=integrity-referential

[12] B. Marr, *Key Performance Indicators (KPI): The 75 Measures Every Manager Needs To Know*. Pearson UK, 2012.

[13] "Create a form in Access - Microsoft Support." Accessed: Mar. 12, 2025. [Online]. Available: https://support.microsoft.com/en-gb/office/create-a-form-in-access-5d550a3d-92e1-4f38-9772-7e7e21e80c6b

[14] "Introduction to reports in Access - Microsoft Support." Accessed: Mar. 12, 2025. [Online]. Available: https://support.microsoft.com/en-gb/office/introduction-to-reports-in-access-e0869f59-7536-4d19-8e05-7158dcd3681c

THANK YOU