

Module: Databases and Application Development

Module code: ACCA5026.

Assignment title: AdventureWorks Cycles' Database Design & Implementation.

Submission date: Thursday, 9th January 2024.

Student name: Shaun James Davies.

Student number: 2317135.

Table of Contents

Introduction	3
1. Requirements & Design	3
1.1 Database Design.....	3
1.1.1 ERD development for Adventure Works Cycles.....	3
1.1.2 Normalised to 3 rd Normal Form (3NF)	3
1.1.3 Appropriate Attributes Demonstrating Design Structure.....	4
1.1.4 Primary and FKs	4
1.2 Database implementation	4
1.2.1 SQL Statements.....	4
1.2.2 Foreign Keys to maintain data consistency	4
1.2.3 Check Constraints	5
1.3 Populating the database	6
1.3.1 Insert functions	6
1.3.2 Testing functionality of the primary key, foreign key, and check constraints.....	9
1.3.2.1 Constraints	9
1.3.2.2 Foreign Key.....	11
1.3.2.3 Primary Key Constraints.....	12
2. Scenario testing database.....	13
2.A Commit	13
2.B Rollback.....	13
3. Performance Tuning Enhancement	14
Conclusion.....	15
References	16
Appendix A.....	17
Appendix B	19

Introduction

A database with appropriate and normalised tables is required for the company AdventureWorks Cycles. This attempt to align this company with industry standard tools such as databases. This innovation project is not only essential in competing with competitors and making company systems easier for users but an efficient system for storing this type of data will improve data retrieval. This data infrastructure needs to provide tables consisting of products and manufacturers through to orders and feedback with processes that connect the business elements like products and manufactures uploading their data to be used within features like order processing, product listing and customer/product feedback.

1. Requirements & Design

An adequate depiction of entity-relationship diagram (ERD) is required before creating the database. This design element wouldn't be possible without the work of P.P. Chen. For AdventureWorks Cycles, the ERD plays "an important role in the... database design" [1]. Importantly, considering different types of attributes like composite, multivalued, stored and others will be crucial in understanding the requirements of the database. This will allow the infrastructure to fit for the business.

1.1 Database Design

1.1.1 ERD development for Adventure Works Cycles

To digest what is required for the company, a simple diagram was constructed as detailed in Figure 1.

Expanding on this thought, a conceptual data model was made designed with a view of "overall structure of data" to understand the "information required to support" AdventureWorks Cycles which can be detailed in Figure 2. It uses basic constructs to detail the problem case [2]. Creating a logical model is the next step which requires more detail and expands from the conceptual [3]. The physical design is the last step of the ERD process as shown in Figure 4 and includes data types and if referential constraints are needed per table [3].

The ERD represents the Adventure Works Cycles database schema, outlining key entities such as Manufacturer, Product, Customer, Order, and Feedback. Relationships are clearly established including ProductCategory for many-to-many links between Product and Category, as well as ProductTags for tagging products with the values in the table Tags. As described prior by the figures, the entirety of the relationships are shown as per the needs of the company.

To include multiple contact details for the manufacturer, a junction table is used between Manufacturer and ContactDetails. To prevent reuse of dimensions the same occurs for the dimensions of products and order details. To prevent redundancy, an OrderStatus table is included as "normalisation requires additional tables" [4].

1.1.2 Normalised to 3rd Normal Form (3NF)

All tables are designed to adhere to the principles of Third Normal Form (3NF), ensuring attributes are solely dependent on their respective Primary Keys (PKs). For instance, attributes like Name and Price in the Product table depend only on ProductID. Extracting all attributes into their own table, normalising the data, reduces the duplication of data, data redundancy and can "maintain atomicity" [5]. While OrderDetails manages many-to-many relationships between Order and Product, storing transactional details such as Quantity and Price. Similarly, Feedback attributes, including Rating and Comments, are tied exclusively to FeedbackID. This approach eliminates redundancy, enhances data integrity, and facilitates efficient querying and updating. Continuing, Thomas, C states that "no non-candidate key attribute is transitively dependent on any candidate key" which for the tables discussed

is true. Normalisation minimises attributes in tables, groups logically dependent ones, and reduces redundancy while using foreign keys (FKs) to connect “related relations” [6] [4].

1.1.3 Appropriate Attributes Demonstrating Design Structure

The database schema includes attributes that are vital for Adventure Works Cycles' operations. For example, the Manufacturer table captures details such as Name, Website, and RegistrationDate, while the Product table includes fields like Name, Price, and StockQuantity. The OrderDetails table stores critical order-related information such as Quantity, Price, and LineTotal, and the Feedback table records Rating, OverallExperience, and ExperienceComments. These attributes ensure the system effectively supports core business processes such as product management, order processing, and customer feedback.

1.1.4 Primary and FKs

Primary and FKs are defined throughout the schema ensuring “minimal redundancy” [6]. PKs, such as ProductID and OrderID, uniquely identify records, while FKs, like CategoryID in ProductCategory and CustomerID in Order, establish relationships between tables. Referential integrity will be maintained as the database is used because it follows the rules set out in fundamentals of Database Systems [7]. These keys facilitate one-to-many and many-to-many relationships that supports Adventure Works Cycles' as set out in their consultation.

1.2 Database implementation

1.2.1 SQL Statements

The database implementation for Adventure Works Cycles focuses on translating the designed schema into a fully functional relational database using SQL. This database structure adheres to normalisation principles (3NF) to eliminate “redundancy” and ensure data consistency [4]. Each table is designed to represent a specific business entity such as Manufacturer, Product, Order, and Feedback, with attributes tailored to meet operational needs.

1.2.2 Foreign Keys to maintain data consistency

PKs are defined for each table to uniquely identify uniquely records and ensure efficient indexing. For example, ProductID in the Product table and OrderID in the Order table act as unique identifiers, allowing reliable referencing and faster query execution. FKs enforce referential integrity by maintaining valid relationships between related tables such as linking Order to Customer via CustomerID. This ensures that no orphan records exist when deleting with a constraint of “ON DELETE CASCADE” like the later code will implement, maintaining data integrity across the database [8].

```
19      -- ManufacturerContactDetails Table
20  ● CREATE TABLE IF NOT EXISTS ManufacturerContactDetails ( -- Same as previous line in previous table
21      ManufacturerID INT NOT NULL, -- Foreign Key to Manufacturer (the Primary Key for the mentioned table is used
22      ContactID INT NOT NULL, -- Foreign Key to ContactDetails (the Primary Key for the mentioned table is used her
23      PRIMARY KEY (ManufacturerID, ContactID), -- Composite Primary Key
24      FOREIGN KEY (ManufacturerID) REFERENCES Manufacturer(ManufacturerID),
25      FOREIGN KEY (ContactID) REFERENCES ContactDetails(ContactID)
26  );
27  --
```

```

50      -- ProductCategory Table
51 • ○ CREATE TABLE IF NOT EXISTS ProductCategory ( -- Same as previous line in previous table
52      ProductID INT, -- Foreign Key to Product
53      CategoryID INT, -- Foreign Key to Category
54      PRIMARY KEY (ProductID, CategoryID), -- Composite Primary Key
55      FOREIGN KEY (ProductID) REFERENCES Product(ProductID),
56      FOREIGN KEY (CategoryID) REFERENCES Category(CategoryID)
57  );

65      -- ProductTags Table
66 • ○ CREATE TABLE IF NOT EXISTS ProductTags ( -- Same as previous line in previous table
67      ProductID INT NOT NULL, -- Foreign Key to Product
68      TagID INT NOT NULL, -- Foreign Key to Tags
69      PRIMARY KEY (ProductID, TagID), -- Composite Primary Key
70      FOREIGN KEY (ProductID) REFERENCES Product(ProductID),
71      FOREIGN KEY (TagID) REFERENCES Tags(TagID)
72  );

```

These tables above include FKs, including ManufacturerContactDetails, Product, ProductCategory, ProductTags, ProductDimensions, Order, OrderDetails, Feedback. Instead of duplication which increases file size, this method allows for data consistency [8].

1.2.3 Check Constraints

To further enforce data integrity, check constraints are implemented to validate specific column values. For instance, a check constraint in the Feedback table ensures that Rating values fall between 1 and 5, while the StockQuantity in the Product table must always be non-negative. These constraints prevent invalid data entry and ensure compliance with business rules.

Furthermore, the SQL code includes comments to explain the purpose of each table, attribute, and constraint, making the implementation clear and easy to maintain by the database administrator. The database design and implementation provide a footing for managing core processes discussed.

```

35      -- Product Table
36 • ○ CREATE TABLE IF NOT EXISTS Product ( -- Same as previous line in previous table
37      ProductID INT AUTO_INCREMENT PRIMARY KEY, -- Unique ID for products
38      ProductName VARCHAR(255) NOT NULL, -- Product name
39      ProductDescription VARCHAR(255), -- Description of what the product is
40      Price DECIMAL(10,2) NOT NULL, -- Product price
41      StockQuantity INT NOT NULL CHECK (StockQuantity >= 0), -- Non-negative stock by using check
42      DiscountedPrice DECIMAL(10,2), -- Discounted price
43      LaunchDate DATE, -- Launch date of said product
44      ManufacturerID INT NOT NULL, -- Foreign Key to Manufacturer
45      CategoryID INT NOT NULL, -- Foreign Key to Category
46      FOREIGN KEY (ManufacturerID) REFERENCES Manufacturer(ManufacturerID),
47      FOREIGN KEY (CategoryID) REFERENCES Category(CategoryID)
48  );

```

This check constraint ensures that the stock level for each product will be positive by using a conditional formula where the check is carried out. The condition is simply greater than or equal to 0.

```

118 -- OrderDetails Table
119 • CREATE TABLE IF NOT EXISTS OrderDetails ( -- Same as previous line in previous table
120     OrderDetailID INT AUTO_INCREMENT PRIMARY KEY, -- Unique ID for order details
121     OrderID INT NOT NULL, -- Foreign Key to Order
122     ProductID INT NOT NULL, -- Foreign Key to Product
123     Quantity INT CHECK (Quantity > 0) NOT NULL, -- The amount of product. Quantity must be positive, the check ensure
124     Price DECIMAL(10,2) NOT NULL, -- Price of the product
125     LineTotal DECIMAL(10,2) AS (Quantity * Price) STORED, -- Calculated line total
126     FOREIGN KEY (OrderID) REFERENCES `Order`(OrderID),
127     FOREIGN KEY (ProductID) REFERENCES Product(ProductID)
128 );

```

Another useful check constraint in the database is within OrderDetails. Similarly to the previous constraint, this checks the value of an attribute using a conditional formula. Here, if Quantity is greater than one, but not zero as the previous, the constraint will allow data to enter into the table.

```

130 -- Feedback Table
131 • CREATE TABLE IF NOT EXISTS Feedback ( -- Same as previous line in previous table
132     FeedbackID INT AUTO_INCREMENT PRIMARY KEY, -- Unique ID for feedback
133     Rating INT NOT NULL CHECK (Rating BETWEEN 1 AND 5), -- This is the scoring the user decides to give. Rating 1-5
134     Comments VARCHAR(255), -- Customer feedback on product by accepting 255 characters of criticism/praise
135     FeedbackDate DATE NOT NULL, -- Feedback date
136     OverallExperience INT CHECK (OverallExperience BETWEEN 1 AND 5), -- Overall experience
137     ExperienceComments VARCHAR(255), -- Customer feedback on experience by accepting 255 characters of criticism/praise
138     ProductID INT NOT NULL, -- Foreign Key to Product
139     CustomerID INT NOT NULL, -- Foreign Key to Customer
140     FOREIGN KEY (ProductID) REFERENCES Product(ProductID),
141     FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)
142 );

```

The data that attempts to enter the database and its tables will not pass into memory in these tables if the checks are not passed.

1.3 Populating the database

1.3.1 Insert functions

To effectively test the database functionality, realistic records will be inserted into each table using SQL INSERT statements. These records will be crafted to align with the schema's relationships and constraints as set out by Figure 3, ensuring the PKs/FKs, and check constraints are validated.

The data will follow a hierarchical approach, starting with independent tables like Manufacturer, Category, and Customer, which do not rely on FKs. Once these base records are established, dependent tables such as Product, Order, and Feedback will be populated, referencing the PKs of related tables to enforce referential integrity [8]. For instance, products will include valid ManufacturerID and CategoryID values, while orders will include valid CustomerID and OrderStatusID.

```

1 -- Manufacturer Table
2 • INSERT INTO Manufacturer (ManufacturerName, RegistrationDate, Address, Website)
3 VALUES
4 ('Shauny Cycles', '2000-11-24', '123 Mayweather, London, SW1A 1AA', 'https://shauncycles.co.uk'),
5 ('Gareth Bikes', '2022-12-01', '45 Gareth Lane, Manchester, M1 111', 'https://garethbikes.co.uk'),
6 ('Gaynorfied', '2022-03-29', '78 Wonderscope, Birmingham, B2 681', NULL),
7 ('Adamtastic', '2025-01-01', '321, Cardiff, CF10 1DD', 'https://adamstastic.co.uk');
8

```

```

9      -- ContactDetails Table
10 •   INSERT INTO ContactDetails (ContactType, ContactValue)
11     VALUES
12     ('Email', 'support@shauncycles.co.uk'),
13     ('Phone', '+44 70770553'),
14     ('Email', 'infoworld@garethbikes.co.uk'),
15     ('Phone', '+44 12345678');
16
17     -- ManufacturerContactDetails Table
18 •   INSERT INTO ManufacturerContactDetails (ManufacturerID, ContactID)
19     VALUES
20     (1, 1),
21     (1, 2),
22     (2, 3),
23     (4, 4);

25     -- Category Table
26 •   INSERT INTO Category (CategoryName, CategoryDescription)
27     VALUES
28     ('Road Bikes', 'High-performance bikes designed for road cycling.'),
29     ('Mountain Bikes', 'Durable bikes built for off-road terrains.'),
30     ('Hybrid Bikes', 'Versatile bikes suitable for both road and trails.'),
31     ('Mystical Bikes', 'The bikes are the hidden gem of the store - magic bikes.');
```

```

32
33     -- Product Table
34 •   INSERT INTO Product (ProductName, ProductDescription, Price, StockQuantity, DiscountedPrice, LaunchDate, ManufacturerID)
35     VALUES
36     ('Masterful 3000', 'A road bike.', 1299.99, 50, 1199.99, '2023-04-01', 1, 1),
37     ('Fail May-ser', 'A mountain bike.', 999.99, 30, NULL, '2023-05-15', 2, 2),
38     ('Scary ether', 'A hybrid bike.', 799.99, 40, 749.99, '2023-06-20', 3, 3),
39     ('Magic Mark', 'A high-quality cycling helmet.', 59.99, 100, 49.99, '2023-07-01', 4, 4);

41     -- ProductCategory Table
42 •   INSERT INTO ProductCategory (ProductID, CategoryID)
43     VALUES
44     (1, 1),
45     (2, 2),
46     (3, 3),
47     (4, 4);
48
49     -- Tags Table
50 •   INSERT INTO Tags (TagName)
51     VALUES
52     ('Lightweight'),
53     ('Durable'),
54     ('Eco-Friendly'),
55     ('Magical');
```

```

56

```

```

57      -- ProductTags Table
58 •   INSERT INTO ProductTags (ProductID, TagID)
59      VALUES
60      (1, 1),
61      (2, 2),
62      (3, 3),
63      (4, 4);

65      -- Dimensions Table
66 •   INSERT INTO Dimensions (ProductLength, ProductWidth, ProductHeight)
67      VALUES
68      (6.50, 9.50, 12.00),
69      (20.00, 7.70, 5.30),
70      (80.50, 100.50, 111.00),
71      (123.90, 120.65, 1201.25);
72
73      -- ProductDimensions Table
74 •   INSERT INTO ProductDimensions (ProductID, DimensionID)
75      VALUES
76      (1, 1),
77      (2, 2),
78      (3, 3),
79      (4, 4);

81      -- Customer Table
82 •   INSERT INTO Customer (CustomerName, PhoneNumber, CustomerPoints)
83      VALUES
84      ('Dafydd Jones', '+44 123 456 7890', 50),
85      ('Carys Williams', '+44 987 654 3210', 100),
86      ('Rhys Morgan', '+44 123 123 4567', 75),
87      ('Sian Evans', '+44 444 444 6543', 60);
88

```

```

1      -- OrderStatus Table
2  •   INSERT INTO OrderStatus (StatusName, OrderStatusDescription)
3      VALUES
4      ('Pending', 'Waiting confirmation.'),
5      ('Confirmed', 'The Order has been confirmed.'),
6      ('Shipped', 'Shipped!!!'),
7      ('Delivered', 'Delivered!!!!');
8
9      -- Order Table
10  •  INSERT INTO `Order` (CustomerID, OrderDate, ShippingDate, OrderStatusID, TotalAmount)
11      VALUES
12      (1, '2023-08-15', '2024-06-21', 3, 1299.99),
13      (2, '2023-08-16', '2024-01-01', 4, 999.99),
14      (3, '2023-08-17', '2025-01-02', 2, 799.99),
15      (4, '2023-08-18', '2025-01-03', 1, 59.99);
16
17      -- OrderDetails Table
18  •  INSERT INTO OrderDetails (OrderID, ProductID, Quantity, Price)
19      VALUES
20      (1, 1, 1, 1299.99),
21      (2, 2, 1, 999.99),
22      (3, 3, 1, 799.99),
23      (4, 4, 1, 59.99);
24
25      -- Feedback Table
26  •  INSERT INTO Feedback (Rating, Comments, FeedbackDate, OverallExperience, ExperienceComments, ProductID, CustomerID)
27      VALUES
28      (5, 'Cynnyrch rhagorol, cadarn iawn!', '2023-08-20', 5, 'Pryniant gorau erioed.', 1, 1),
29      (4, 'Ansawdd da ond ychydig yn ddud.', '2023-08-21', 4, 'Bodddhaol iawn.', 2, 2),
30      (3, 'Profiad cyffredin.', '2023-08-22', 3, 'Gallai fod yn well.', 3, 3),
31      (5, 'Amazingly lightweight, the flight was the best part!', '2023-08-23', 5, 'Terrific :).', 4, 4);
32

```

158	18:11:20	CREATE TABLE IF NOT EXISTS Feedback (-- Same as previous line in previous table	Fe...	0 row(s) affected		0.125 sec
159	18:55:05	INSERT INTO Manufacturer (ManufacturerName, RegistrationDate, Address, Website) VALU...		4 row(s) affected Records: 4 Duplicates: 0 Warnings: 0		0.125 sec
160	18:57:53	INSERT INTO ContactDetails (ContactType, ContactValue) VALUES ('Email', 'support@shau...		4 row(s) affected Records: 4 Duplicates: 0 Warnings: 0		0.016 sec
161	18:57:53	INSERT INTO ManufacturerContactDetails (ManufacturerID, ContactID) VALUES (1, 1), (1, 2...		4 row(s) affected Records: 4 Duplicates: 0 Warnings: 0		0.031 sec
162	19:00:25	INSERT INTO Category (CategoryName, CategoryDescription) VALUES ('Road Bikes', 'High...		4 row(s) affected Records: 4 Duplicates: 0 Warnings: 0		0.016 sec
163	19:00:25	INSERT INTO Product (ProductName, ProductDescription, Price, StockQuantity, Discounted...		4 row(s) affected Records: 4 Duplicates: 0 Warnings: 0		0.078 sec
164	19:02:13	INSERT INTO ProductCategory (ProductID, CategoryID) VALUES (1, 1), (2, 2), (3, 3), (4, 4)		4 row(s) affected Records: 4 Duplicates: 0 Warnings: 0		0.016 sec
165	19:02:13	INSERT INTO Tags (TagName) VALUES ('Lightweight'), ('Durable'), ('Eco-Friendly'), ('Magical')		4 row(s) affected Records: 4 Duplicates: 0 Warnings: 0		0.000 sec
166	19:02:13	INSERT INTO ProductTags (ProductID, TagID) VALUES (1, 1), (2, 2), (3, 3), (4, 4)		4 row(s) affected Records: 4 Duplicates: 0 Warnings: 0		0.015 sec
167	19:06:30	INSERT INTO Dimensions (ProductLength, ProductWidth, ProductHeight) VALUES (6.50, 9...		4 row(s) affected Records: 4 Duplicates: 0 Warnings: 0		0.016 sec
168	19:06:30	INSERT INTO ProductDimensions (ProductID, DimensionID) VALUES (1, 1), (2, 2), (3, 3), (4, 4)		4 row(s) affected Records: 4 Duplicates: 0 Warnings: 0		0.015 sec
169	19:06:30	INSERT INTO Customer (CustomerName, PhoneNumber, CustomerPoints) VALUES ('Dafydd ...		4 row(s) affected Records: 4 Duplicates: 0 Warnings: 0		0.016 sec
170	19:08:29	INSERT INTO OrderStatus (StatusName, OrderStatusDescription) VALUES ('Pending', 'Wati...		4 row(s) affected Records: 4 Duplicates: 0 Warnings: 0		0.016 sec
171	19:08:29	INSERT INTO `Order` (CustomerID, OrderDate, ShippingDate, OrderStatusID, TotalAmount) V...		4 row(s) affected Records: 4 Duplicates: 0 Warnings: 0		0.015 sec
172	19:08:29	INSERT INTO OrderDetails (OrderID, ProductID, Quantity, Price) VALUES (1, 1, 1, 1299.99), ...		4 row(s) affected Records: 4 Duplicates: 0 Warnings: 0		0.016 sec
173	19:08:29	INSERT INTO Feedback (Rating, Comments, FeedbackDate, OverallExperience, Experience...		4 row(s) affected Records: 4 Duplicates: 0 Warnings: 0		0.031 sec

The output shows all inserts are appropriate when entered and do not contradict any constraints or operators.

1.3.2 Testing functionality of the primary key, foreign key, and check constraints

1.3.2.1 Constraints

1) Not Null

a) Entering null ManufacturerName

```
1 -- Not Null Constraints test
2 -- Manufacturer Table
3 -- This test attempts to insert a row into manufacturer with the ManufacturerName NULL.
4 • INSERT INTO Manufacturer (ManufacturerName, RegistrationDate, Address, Website)
5 VALUES (NULL, '2023-09-01', '123 xxx, London, SW1C 1CC', 'https://xxx.co.uk');
6 -- This test, if the code is correct, should fail because ManufacturerName cannot be NULL.

4 • CREATE TABLE IF NOT EXISTS Manufacturer ( -- Creates the table if not already present
5     ManufacturerID INT AUTO_INCREMENT PRIMARY KEY, -- Unique ID for manufacturers, auto increments
6     ManufacturerName VARCHAR(255) NOT NULL, -- Manufacturer's name. Not null means the attribute value cannot be empty
7     RegistrationDate DATE NOT NULL, -- Date (formatted as a date) of registration onto AdventureWorks Cycles database
8     Address VARCHAR(255), -- Address of the manufacturer
9     Website VARCHAR(255) -- Manufacturer's website
10 );
```

Output				
Action Output				
#	Time	Action	Message	Duration / Fetch
173	19:08:29	INSERT INTO Feedback (Rating, Comments, FeedbackDate, OverallExperience, Experience...	4 row(s) affected Records: 4 Duplicates: 0 Warnings: 0	0.031 sec
174	19:16:31	INSERT INTO Manufacturer (ManufacturerName, RegistrationDate, Address, Website) VALU...	Error Code: 1048. Column 'ManufacturerName' cannot be null	0.031 sec

This attempts to insert an entry with a ManufacturerName that is against attribute constraint. The output, ergo, prevented.

```
1 -- Customer Table
2 -- This test tries to enter a customername as null which is against the constraint.
3 • INSERT INTO Customer (CustomerName, PhoneNumber, CustomerPoints)
4 VALUES (NULL, '+44 123 456 7890', 10);
5 -- -- This test, if the code is correct, should fail because CustomerName cannot be NULL.
```

Output				
Action Output				
#	Time	Action	Message	Duration / Fetch
174	19:16:31	INSERT INTO Manufacturer (ManufacturerName, RegistrationDate, Address, Website) VALU...	Error Code: 1048. Column 'ManufacturerName' cannot be null	0.031 sec
175	19:18:02	INSERT INTO Customer (CustomerName, PhoneNumber, CustomerPoints) VALUES (NULL, '+...	Error Code: 1048. Column 'CustomerName' cannot be null	0.000 sec

This is the same constraint test on a different table with the same output.

2) Default values

```
1 -- Default Values
2 -- Customer Table
3 -- This test tries the scenario of entering no value for CustomerPoints.
4 • INSERT INTO Customer (CustomerName, PhoneNumber)
5 VALUES ('xxxx', '+44 777 123 4567');
6 -- -- This test, if the code is correct, should insert CustomerPoints as 0 by default.
```

As the Customer table code dictates, CustomerPoints should enter as 0 as a default value. When using the above code, a CustomerPoints isn't stated and should enter as previously described.

Output				
Action Output				
#	Time	Action	Message	Duration / Fetch
175	19:18:02	INSERT INTO Customer (CustomerName, PhoneNumber, CustomerPoints) VALUES (NULL, '+...	Error Code: 1048. Column 'CustomerName' cannot be null	0.000 sec
176	19:20:06	INSERT INTO Customer (CustomerName, PhoneNumber) VALUES ('xxxx', '+44 777 123 4567')	1 row(s) affected	0.015 sec

Insert statement passed.

```
1 • SELECT CustomerName, CustomerPoints FROM Customer;
```

CustomerName	CustomerPoints
Dafydd Jones	50
Carys Williams	100
Rhys Morgan	75
Sian Evans	60
xxxx	0

The above picture details that the INSERT entered our test in as expected; CustomerPoints equals 0.

3) Cascading delete

```
1 -- Cascading Deletes
2 -- Delete Manufacturer and Check Cascade in Product Table
3 -- This test tries to delete Manufacturer with an ID of 1 to test if the update cascades through to product
4 • DELETE FROM Manufacturer WHERE ManufacturerID = 1;
5 -- -- This test, if the code is correct, should delete all products linked to ManufacturerID 1 in the Product table.
6
```

This will test if an entry can be deleted based on primary key.

178 19:23:28 DELETE FROM Manufacturer WHERE ManufacturerID = 1 Error Code: 1451. Cannot delete or update a parent row: a foreign key constraint fails ('datab... 0.047 sec

The code failed as the ON DELETE CASCADE with no present within the database.

```
50 -- Product Table (with on delete cascade)
51 • CREATE TABLE IF NOT EXISTS Product ( -- Same as previous line in previous table
52     ProductID INT AUTO_INCREMENT PRIMARY KEY, -- Unique ID for products
53     ProductName VARCHAR(255) NOT NULL, -- Product name
54     ProductDescription VARCHAR(255), -- Description of what the product is
55     Price DECIMAL(10,2) NOT NULL, -- Product price
56     StockQuantity INT NOT NULL CHECK (StockQuantity >= 0), -- Non-negative stock by using check
57     DiscountedPrice DECIMAL(10,2), -- Discounted price
58     LaunchDate DATE, -- Launch date of said product
59     ManufacturerID INT NOT NULL, -- Foreign Key to Manufacturer
60     CategoryID INT NOT NULL, -- Foreign Key to Category
61     FOREIGN KEY (ManufacturerID) REFERENCES Manufacturer(ManufacturerID) ON DELETE CASCADE,
62     FOREIGN KEY (CategoryID) REFERENCES Category(CategoryID) ON DELETE CASCADE
63 );
```

By editing all FKs with the aforementioned constraint, the delete function can occur without leaving orphaned data.

228 19:52:12 DELETE FROM Manufacturer WHERE ManufacturerID = 1 1 row(s) affected 0.016 sec

Outcome changed.

1.3.2.2 Foreign Key

1) A insert with a primary key that doesn't exist.

```

1  -- Foreign Key Constraints
2  -- Product Table
3  -- This test tries to insert a product with a ManufacturerID 12345
4  • INSERT INTO Product (ProductName, ProductDescription, Price, StockQuantity, DiscountedPrice, LaunchDate, ManufacturerID,
5  VALUES ('xxxx', 'xxxx', 100.00, 10, NULL, '2023-10-01', 12345, 1);
6  -- -- This test, if the code is correct, should fail because ManufacturerID 12345 does not exist in Manufacturer table.
7  |

```

230 19:55:25 INSERT INTO Product (ProductName, ProductDescription, Price, StockQuantity, Discounte... Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails ('adventure... 0.031 sec

The test attempts to enter data with a non-existent ManufacturerID. The output fails to be produced.

```

1  -- ManufacturerContactDetails Table
2  -- This test uses ManufacturerID 12345 again to see if there's any contact details for this non existant Manufacturer
3  • INSERT INTO ManufacturerContactDetails (ManufacturerID, ContactID)
4  VALUES (12345, 1);
5  -- -- This test, if the code is correct, should fail because ManufacturerID 12345 does not exist.
6  |

```

231 19:57:29 INSERT INTO ManufacturerContactDetails (ManufacturerID, ContactID) VALUES (12345, 1) Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails ('adventure... 0.000 sec

This is identical in purpose as above.

2) Negative value.

```

1  -- Check Constraints
2  -- Product Table (StockQuantity >= 0)
3  -- This test is trying to enter negative stock which shouldn't be allowed.
4  INSERT INTO Product (ProductName, ProductDescription, Price, StockQuantity, DiscountedPrice, LaunchDate, ManufacturerID,
5  VALUES ('xxxxxxxx', 'xxxxxxx', 200.00, -5, NULL, '2019-10-01', 1, 1);
6  -- -- This test, if the code is correct, should fail because StockQuantity cannot be negative.
7  |

```

234 20:05:44 INSERT INTO Product (ProductName, ProductDescription, Price, StockQuantity, Discounte... Error Code: 3819. Check constraint 'product_chk_1' is violated. 0.016 sec

Within Product, StockQuantity has a condition to be created; greater than or equal to 0. This is being contested by the above insert.

1.3.2.3 Primary Key Constraints

1)

```

1  -- Primary Key and Auto-Increment Constraints
2  -- Manufacturer Table (Auto-Increment and Primary Key)
3  -- This test tries to make a ManufacturerID of 1 which already exists and shouldn't be allowed.
4  • INSERT INTO Manufacturer (ManufacturerID, ManufacturerName, RegistrationDate, Address, Website)
5  VALUES (1, 'xxxx', '2019-01-01', '456 Shaun Street, London, SHA! 1BB', 'https://xxxx.co.uk');
6  -- -- This test, if the code is correct, should fail because ManufacturerID 1 already exists.
7  |
8  |

```

233 20:03:00 INSERT INTO Manufacturer (ManufacturerID, ManufacturerName, RegistrationDate, Address... Error Code: 1062. Duplicate entry '1' for key 'manufacturer.PRIMARY' 0.000 sec

2)

```

1  -- Composite Primary Key Constraints
2  -- ManufacturerContactDetails Table
3  -- This test is trying to add a ManufacturerID/ ContactID pairing that already exists which shouldn't be allowed
4  INSERT INTO ManufacturerContactDetails (ManufacturerID, ContactID)
5  VALUES (1, 1);
6  -- -- This test, if the code is correct, should fail because the combination of ManufacturerID and ContactID already exists
7

```

✖ 236 20:07:04 INSERT INTO ManufacturerContactDetails (ManufacturerID, ContactID) VALUES (1, 1) Error Code: 1062. Duplicate entry '1-1' for key 'manufacturercontactdetails.PRIMARY' 0.000 sec

ManufacturerID of both already exists therefore the tests fail as duplicate entries cannot be made for PKs.

2. Scenario testing database

2.A Commit

In the first scenario of placing an order, a customer places an order for two products. The transaction involves inserting order details into the Order table, adding purchased items to the OrderDetails table, reducing stock quantities in the Product table, and updating the customer's points in the Customer table. The rationale behind suggesting this commit is this proposes a common scenario that can occur in any business and especially one like AdventureWorks Cycles. As Bernstein and Newcomer (2009) state, “the transaction program is all-or-nothing” [9], ensuring consistency only after a successful commit. By finalising all changes through the COMMIT operation, the database guarantees accurate inventory updates and loyalty point calculations.

```

1  -- Commit
2  • START TRANSACTION; -- Begin the transaction in order to place an order
3
4  -- Insert a new order
5  • INSERT INTO `Order` (CustomerID, OrderDate, ShippingDate, OrderStatusID, TotalAmount)
6  VALUES (1, '2023-11-24', '2023-12-24', 2, 1999.98);
7
8  -- Get the last inserted OrderID for setting up and connecting with OrderDetails
9  • SET @OrderID = LAST_INSERT_ID();
10
11 -- Insert products into OrderDetails as per the diagram detailing relationship
12 • INSERT INTO OrderDetails (OrderID, ProductID, Quantity, Price)
13 VALUES (@OrderID, 1, 1, 1299.99), -- Product 1
14         (@OrderID, 2, 1, 699.99); -- Product 2
15
16 -- Update stock quantities for purchased products: this allows for accurate measurement of stock

```

✓	252	20:34:13	START TRANSACTION	0 row(s) affected
✓	253	20:34:13	INSERT INTO `Order` (CustomerID, OrderDate, ShippingDate, OrderStatusID, TotalAmount) ...	1 row(s) affected
✓	254	20:34:13	SET @OrderID = LAST_INSERT_ID()	0 row(s) affected
✓	255	20:34:13	INSERT INTO OrderDetails (OrderID, ProductID, Quantity, Price) VALUES (@OrderID, 1, 1, ...	2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0
✓	256	20:34:13	UPDATE Product SET StockQuantity = StockQuantity - 1 WHERE ProductID IN (1, 2)	2 row(s) affected Rows matched: 2 Changed: 2 Warnings: 0
✓	257	20:34:13	UPDATE Customer SET CustomerPoints = CustomerPoints + 20 WHERE CustomerID = 1	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0
✓	258	20:34:13	COMMIT	0 row(s) affected

2.B Rollback

The next scenario is a stock availability failure. The rationale behind this is simple. The company may have stock that is insufficient for the order and needs to be cancelled or stopped before it is carried

out. A customer attempts to place an order, but one of the requested products has insufficient stock. The transaction begins with inserting order details into the Order table and adding items to the OrderDetails table. However, during the process of updating stock quantities in the Product table, a check reveals insufficient stock for one or more products. As a result, the transaction is rolled back to cancel all operations. The ROLLBACK operation ensures that no partial changes are saved, preserving the integrity of the database and preventing inconsistent data updates concurred by the article by Won-Young Kim et al that states “transaction rollback is a mechanism for cancelling the effect of operations executed within a database transaction” [10].

```

1  DELIMITER //
2
3  ● CREATE PROCEDURE PlaceOrderWithStockCheck(
4      IN customer_id INT,
5      IN product_id INT,
6      IN quantity INT,
7      IN order_date DATE,
8      IN shipping_date DATE,
9      IN order_status INT,
10     IN total_amount DECIMAL(10,2)
11 )
12 ● BEGIN
13     DECLARE stock_available INT;
14
15     -- Start the transaction
16     START TRANSACTION;
17
18     -- Check stock availability
19     SELECT StockQuantity INTO stock_available

```

259	21:53:19	CREATE PROCEDURE PlaceOrderWithStockCheck(IN customer_id INT, IN product_id INT, IN quantity INT, IN order_date DATE, IN shipping_date DATE, IN order_status INT, IN total_amount DECIMAL(10,2))	0 row(s) affected	0.172 sec
260	21:53:19	Example Execution: CALL PlaceOrderWithStockCheck(2, 3, 2, '2023-12-02', '2023-12-06', ...)	0 row(s) affected	0.063 sec

3. Performance Tuning Enhancement

```

1  -- Indexes for Product Searches
2  ● CREATE INDEX idx_ProductName ON Product (ProductName);
3  -- Speeds up searches by product name select for index. The product name is the main attribute across the platform
4  ● CREATE INDEX idx_CategoryID ON Product (CategoryID);
5  -- Optimises filtering by category as users will likely use this option regularly
6
7  -- Indexes for Order Lookups
8  ● CREATE INDEX idx_CustomerID ON `Order` (CustomerID);
9  -- looking up by customer seems to be a useful tool in any business, this will be useful
10 ● CREATE INDEX idx_OrderDate ON `Order` (OrderDate);
11 -- This will be useful in order to filter efficiently on a popular attribute; OrderDate
12

```

261	21:59:39	CREATE INDEX idx_ProductName ON Product (ProductName)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.407 sec
262	21:59:40	CREATE INDEX idx_CategoryID ON Product (CategoryID)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.093 sec
263	21:59:40	CREATE INDEX idx_CustomerID ON `Order` (CustomerID)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.063 sec
264	21:59:40	CREATE INDEX idx_OrderDate ON `Order` (OrderDate)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.078 sec

To increase performance and user experience it's important to identify key areas of the database. Indexes are code segment that increases speed of retrieval. The following functions are the most

important of the database: product/category/customer and order date recall which benefit from performance enhancement. The indexes have been created as above indicate.

Conclusion

The AdventureWorks Cycles company can now ensure that the requirements of their data infrastructure are met. Practical constraints, rollback and commit functions have been implemented as based on the ERD design process. From product database elements to feedback elements, the appropriate measures have been taken to ensure what essential features have been added. Data integrity has been described throughout as being implemented with referential integrity at the forefront of each stage of the design and implementation. Foreign keys and normalisation are an example of this which create a symbiosis between accurate data, the database and the company. The influence of indexing, especially the secondary indexing, allows for faster computation times that increase performance of the database. By using the rollback and commit features of the database design transactions will become safe and reliable for the queries that the business need and use. As these features have been implemented the company can easily promote scalability as the company grows with increased functionality with queries implemented.

References

- [1] G. Chen, "Conceptual Modeling with the Entity-Relationship Model," in *Fuzzy Logic in Data Modeling*, Boston, MA, Springer, 1998, pp. 19-33.
- [2] S. Al-Fedaghi, "Conceptual Data Modeling: Entity-Relationship Models as Thinging Machines," *International Journal of Computer Science and Network Security*, vol. 21, no. No.9, pp. 147-148, 2021.
- [3] T. Connolly, "Methodology - Logical," in *Database systems: a practical approach to design, implementation and management*, Pearson Education, 2015, pp. 528-563.
- [4] "A Novel Normalization Forms for Relational Database Design," Makkah, Saudi Arabia, Department of Computer Science, Umm Al-Qura University, 2017, pp. 66-70.
- [5] D. S. K. A. Kunal Kumar, "Database Normalization Design Pattern," in *2017 4th IEEE Uttar Pradesh Section International Conference on Electrical, Computer and Electronics (UPCON)*, GLA University, 2017, p. 318.
- [6] T. Connolly, "Normalization," in *Database systems: a practical approach to design, implementation and management*, Pearson Education, 2015, pp. 453-474.
- [7] R. E. Sham Navathe, "Integrity, Referential Integrity,," in *Fundamentals of database systems*, Pearson Addison-Wesley, 2007, p. 102.
- [8] M. S. R. P. Coronel Carlos, "Database Systems," in *Design, implementation and management*, 2010, pp. 66-89.
- [9] E. N. Philip A. Bernstein, "Chapter 1 - Introduction," in *Principles of Transaction Processing*, Elsevier, 2009, p. 10.
- [10] B. S. L. K.-Y. W. Won-Young Kim, "Partial rollback in object-oriented/object-relational database management systems with dual buffer," *Information and Software Technology*, vol. 48, no. 2, p. 121, 2006.

Appendix A

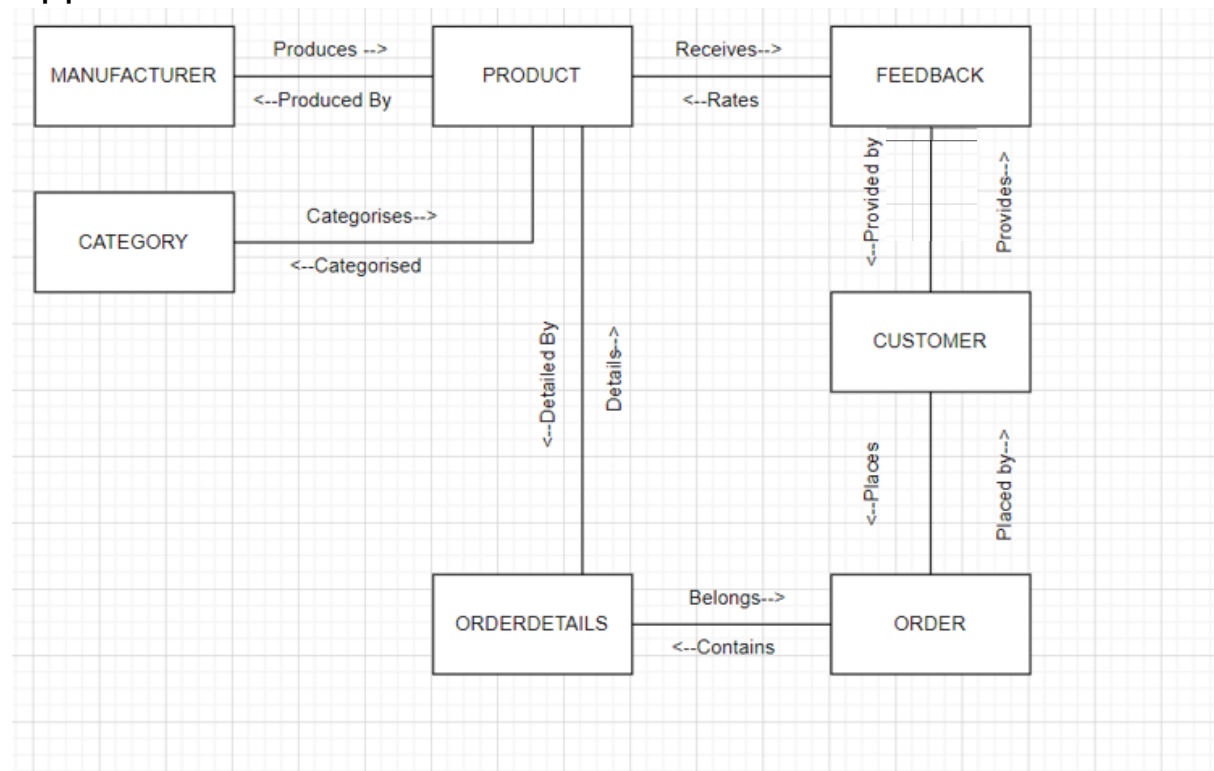


Figure 1: Brainstorm of AdventureWorks Cycles ERD

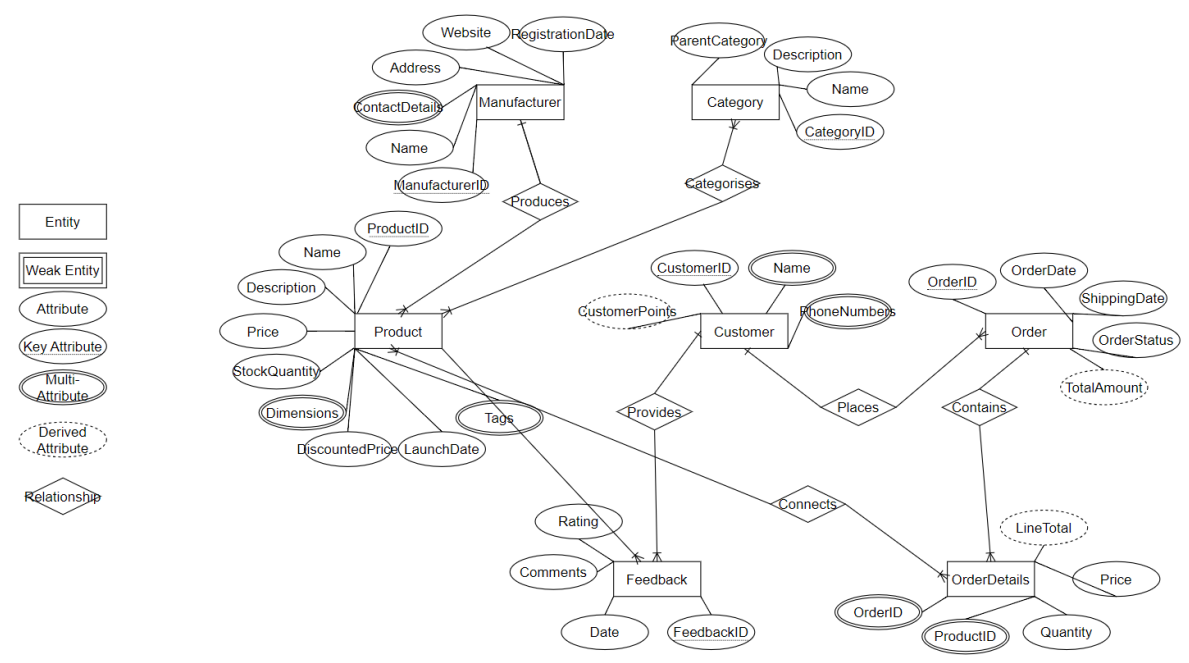


Figure 2: Conceptual data model

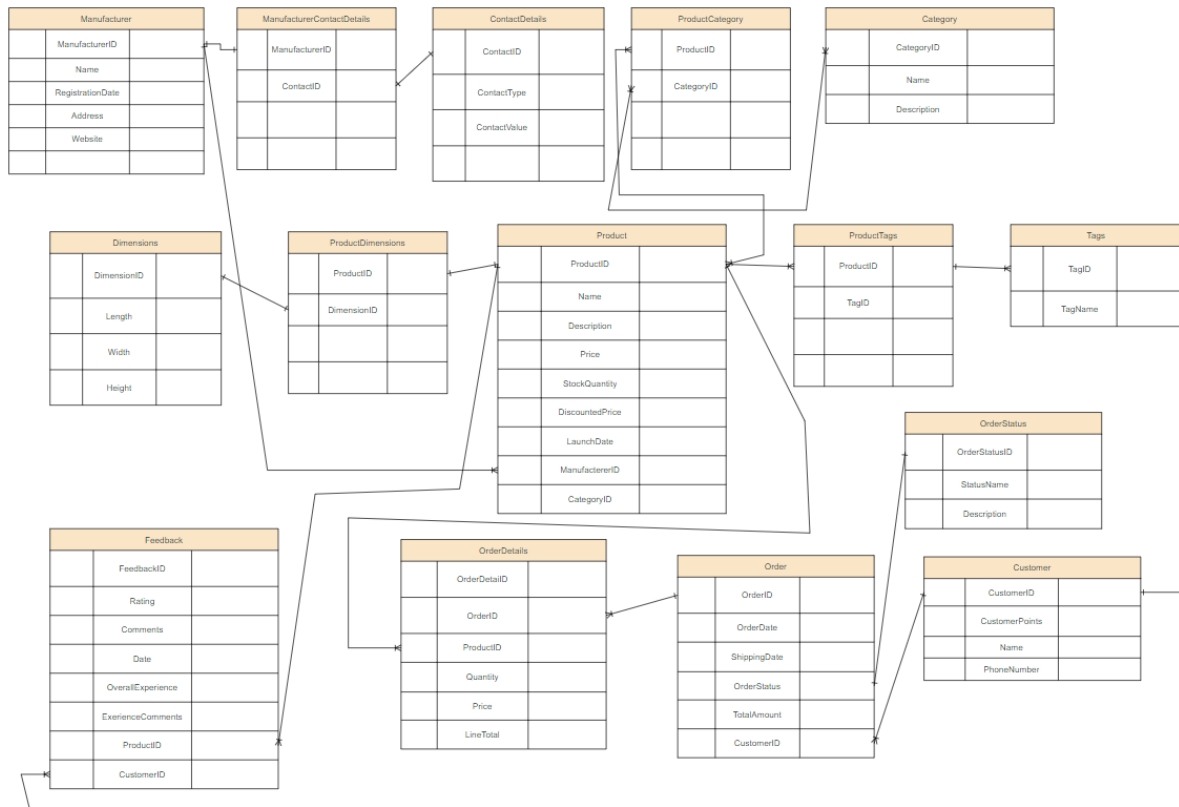


Figure 3: Logical data model

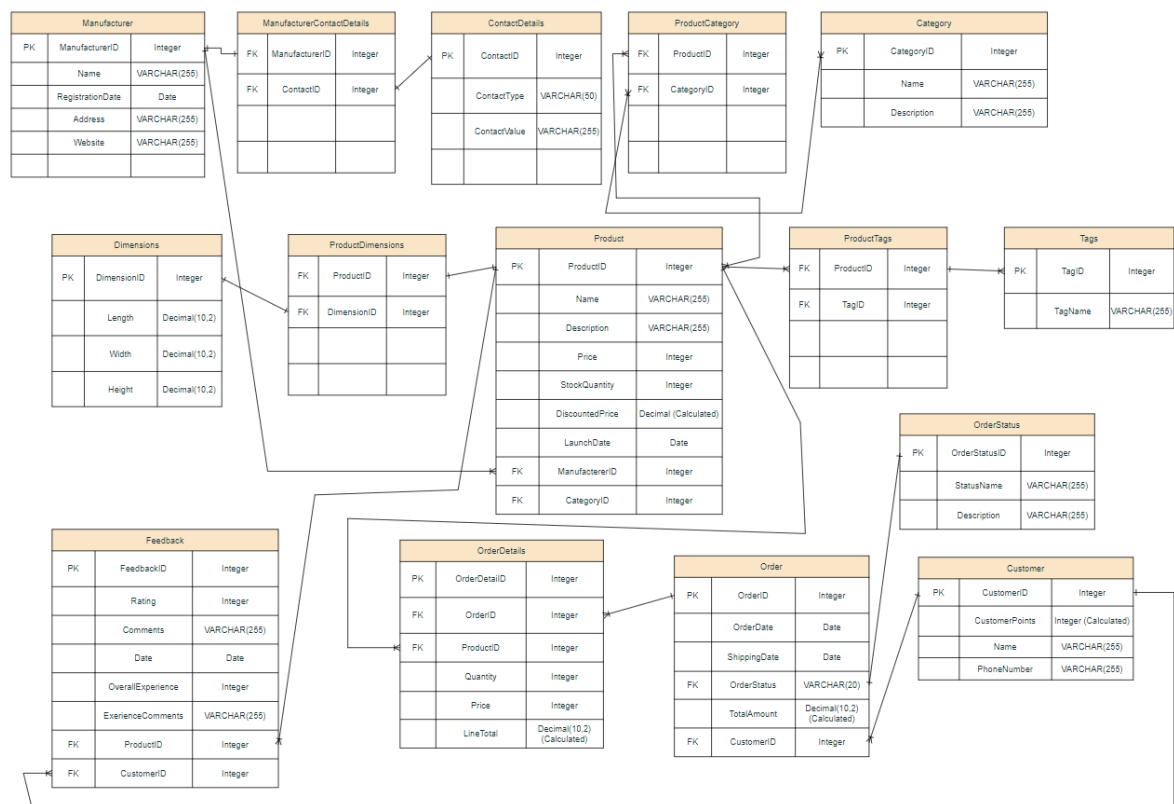


Figure 4: Physical data model

Appendix B

Code

-- Please update line if the database you use is named something else

USE adventureworks_cycles;

-- Manufacturer Table

CREATE TABLE IF NOT EXISTS Manufacturer (-- Creates the table if not already present

ManufacturerID INT AUTO_INCREMENT PRIMARY KEY, -- Unique ID for manufacturers, auto increments

ManufacturerName VARCHAR(255) NOT NULL, -- Manufacturer's name. Not null means the attribute value cannot be empty

RegistrationDate DATE NOT NULL, -- Date (formatted as a date) of registration onto AdventureWorks Cycles database

Address VARCHAR(255), -- Address of the manufacturer

Website VARCHAR(255) -- Manufacturer's website

);

-- ContactDetails Table

CREATE TABLE IF NOT EXISTS ContactDetails (-- Same as previous line in previous table

ContactID INT AUTO_INCREMENT PRIMARY KEY, -- Unique ID for contact details

ContactType VARCHAR(50), -- Type of contact (e.g., any contact type but typically email, phone.)

ContactValue VARCHAR(255) -- The data associated with the contact type (e.g, a phone number for the type phone)

);

-- ManufacturerContactDetails Table

CREATE TABLE IF NOT EXISTS ManufacturerContactDetails (-- Same as previous line in previous table

ManufacturerID INT NOT NULL, -- Foreign Key to Manufacturer (the Primary Key for the mentioned table is used here again as a Foreign Key)

ContactID INT NOT NULL, -- Foreign Key to ContactDetails (the Primary Key for the mentioned table is used here again as a Foreign Key)

PRIMARY KEY (ManufacturerID, ContactID), -- Composite Primary Key

FOREIGN KEY (ManufacturerID) REFERENCES Manufacturer(ManufacturerID) ON DELETE CASCADE,

FOREIGN KEY (ContactID) REFERENCES ContactDetails(ContactID) ON DELETE CASCADE

);

-- Category Table

CREATE TABLE IF NOT EXISTS Category (-- Same as previous line in previous table

CategoryID INT AUTO_INCREMENT PRIMARY KEY, -- Unique ID for categories

CategoryName VARCHAR(255) NOT NULL, -- Category name

CategoryDescription VARCHAR(255) -- Description of what the category is

);

-- Product Table (without on delete cascade)

-- CREATE TABLE IF NOT EXISTS Product (-- Same as previous line in previous table

-- ProductID INT AUTO_INCREMENT PRIMARY KEY, -- Unique ID for products

-- ProductName VARCHAR(255) NOT NULL, -- Product name

-- ProductDescription VARCHAR(255), -- Description of what the product is

-- Price DECIMAL(10,2) NOT NULL, -- Product price

-- StockQuantity INT NOT NULL CHECK (StockQuantity >= 0), -- Non-negative stock by using check

-- DiscountedPrice DECIMAL(10,2), -- Discounted price

-- LaunchDate DATE, -- Launch date of said product

-- ManufacturerID INT NOT NULL, -- Foreign Key to Manufacturer

-- CategoryID INT NOT NULL, -- Foreign Key to Category

-- FOREIGN KEY (ManufacturerID) REFERENCES Manufacturer(ManufacturerID),

-- FOREIGN KEY (CategoryID) REFERENCES Category(CategoryID)

--);

-- Product Table (with on delete cascade)

CREATE TABLE IF NOT EXISTS Product (-- Same as previous line in previous table

ProductID INT AUTO_INCREMENT PRIMARY KEY, -- Unique ID for products

ProductName VARCHAR(255) NOT NULL, -- Product name

ProductDescription VARCHAR(255), -- Description of what the product is

Price DECIMAL(10,2) NOT NULL, -- Product price

StockQuantity INT NOT NULL CHECK (StockQuantity >= 0), -- Non-negative stock by using check

DiscountedPrice DECIMAL(10,2), -- Discounted price

```
LaunchDate DATE, -- Launch date of said product
ManufacturerID INT NOT NULL, -- Foreign Key to Manufacturer
CategoryID INT NOT NULL, -- Foreign Key to Category
FOREIGN KEY (ManufacturerID) REFERENCES Manufacturer(ManufacturerID) ON DELETE CASCADE,
FOREIGN KEY (CategoryID) REFERENCES Category(CategoryID) ON DELETE CASCADE
);
```

-- ProductCategory Table

```
CREATE TABLE IF NOT EXISTS ProductCategory ( -- Same as previous line in previous table
    ProductID INT, -- Foreign Key to Product
    CategoryID INT, -- Foreign Key to Category
    PRIMARY KEY (ProductID, CategoryID), -- Composite Primary Key
    FOREIGN KEY (ProductID) REFERENCES Product(ProductID) ON DELETE CASCADE,
    FOREIGN KEY (CategoryID) REFERENCES Category(CategoryID) ON DELETE CASCADE
);
```

-- Tags Table

```
CREATE TABLE IF NOT EXISTS Tags ( -- Same as previous line in previous table
    TagID INT AUTO_INCREMENT PRIMARY KEY, -- Unique ID for tags
    TagName VARCHAR(255) NOT NULL -- Name of the tag
);
```

-- ProductTags Table

```
CREATE TABLE IF NOT EXISTS ProductTags ( -- Same as previous line in previous table
    ProductID INT NOT NULL, -- Foreign Key to Product
    TagID INT NOT NULL, -- Foreign Key to Tags
    PRIMARY KEY (ProductID, TagID), -- Composite Primary Key
    FOREIGN KEY (ProductID) REFERENCES Product(ProductID) ON DELETE CASCADE,
    FOREIGN KEY (TagID) REFERENCES Tags(TagID) ON DELETE CASCADE
);
```

-- Dimensions Table

CREATE TABLE IF NOT EXISTS Dimensions (-- Same as previous line in previous table

DimensionID INT AUTO_INCREMENT PRIMARY KEY, -- Unique ID for dimensions

ProductLength DECIMAL(10,2), -- Length of the product.

ProductWidth DECIMAL(10,2), -- Width of the product

ProductHeight DECIMAL(10,2) -- Height of the product

);

-- ProductDimensions Table

CREATE TABLE IF NOT EXISTS ProductDimensions (-- Same as previous line in previous table

ProductID INT NOT NULL, -- Foreign Key to Product

DimensionID INT NOT NULL, -- Foreign Key to Dimensions

PRIMARY KEY (ProductID, DimensionID), -- Composite Primary Key

FOREIGN KEY (ProductID) REFERENCES Product(ProductID) ON DELETE CASCADE,

FOREIGN KEY (DimensionID) REFERENCES Dimensions(DimensionID) ON DELETE CASCADE

);

-- Customer Table

CREATE TABLE IF NOT EXISTS Customer (-- Same as previous line in previous table

CustomerID INT AUTO_INCREMENT PRIMARY KEY, -- Unique ID for customers

CustomerName VARCHAR(255) NOT NULL, -- Customer's name

PhoneNumber VARCHAR(20), -- Customer's phone number

CustomerPoints INT DEFAULT 0 -- Customers can gain points by using this attribute

);

-- OrderStatus Table

CREATE TABLE IF NOT EXISTS OrderStatus (-- Same as previous line in previous table

OrderStatusID INT AUTO_INCREMENT PRIMARY KEY, -- Unique ID for each order status

StatusName VARCHAR(255) NOT NULL, -- Name of the status

OrderStatusDescription VARCHAR(255) -- Description of the status

);

-- Order Table

```
CREATE TABLE IF NOT EXISTS `Order` ( -- Same as previous line in previous table
    OrderID INT AUTO_INCREMENT PRIMARY KEY, -- Unique ID for orders
    CustomerID INT NOT NULL, -- Foreign Key to Customer
    OrderDate DATE NOT NULL, -- Date of the order
    ShippingDate DATE, -- Shipping date
    OrderStatusID INT NOT NULL, -- Foreign Key to OrderStatus
    TotalAmount DECIMAL(10,2) NOT NULL, -- Total amount for the order
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID) ON DELETE CASCADE,
    FOREIGN KEY (OrderStatusID) REFERENCES OrderStatus(OrderStatusID) ON DELETE CASCADE
);
```

-- OrderDetails Table

```
CREATE TABLE IF NOT EXISTS OrderDetails ( -- Same as previous line in previous table
    OrderDetailID INT AUTO_INCREMENT PRIMARY KEY, -- Unique ID for order details
    OrderID INT NOT NULL, -- Foreign Key to Order
    ProductID INT NOT NULL, -- Foreign Key to Product
    Quantity INT CHECK (Quantity > 0) NOT NULL, -- The amount of product. Quantity must be positive,
    the check ensures this.
    Price DECIMAL(10,2) NOT NULL, -- Price of the product
    LineTotal DECIMAL(10,2) AS (Quantity * Price) STORED, -- Calculated line total
    FOREIGN KEY (OrderID) REFERENCES `Order`(OrderID) ON DELETE CASCADE,
    FOREIGN KEY (ProductID) REFERENCES Product(ProductID) ON DELETE CASCADE
);
```

-- Feedback Table

```
CREATE TABLE IF NOT EXISTS Feedback ( -- Same as previous line in previous table
    FeedbackID INT AUTO_INCREMENT PRIMARY KEY, -- Unique ID for feedback
    Rating INT NOT NULL CHECK (Rating BETWEEN 1 AND 5), -- This is the scoring the user decides to
    give. Rating 1-5
    Comments VARCHAR(255), -- Customer feedback on product by accepting 255 characters of
    criticism/praise
```

```
FeedbackDate DATE NOT NULL, -- Feedback date

OverallExperience INT CHECK (OverallExperience BETWEEN 1 AND 5), -- Overall experience

ExperienceComments VARCHAR(255), -- Customer feedback on experience by accepting 255
characters of criticism/praise

ProductID INT NOT NULL, -- Foreign Key to Product

CustomerID INT NOT NULL, -- Foreign Key to Customer

FOREIGN KEY (ProductID) REFERENCES Product(ProductID) ON DELETE CASCADE,
FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID) ON DELETE CASCADE
);
```

-- Manufacturer Table

```
INSERT INTO Manufacturer (ManufacturerName, RegistrationDate, Address, Website)
VALUES
('Shauny Cycles', '2000-11-24', '123 Mayweather, London, SW1A 1AA', 'https://shauncycles.co.uk'),
('Gareth Bikes', '2022-12-01', '45 Gareth Lane, Manchester, M1 111', 'https://garethbikes.co.uk'),
('Gaynorfied', '2022-03-29', '78 Wonderscope, Birmingham, B2 G81', NULL),
('Adamtastic', '2025-01-01', '321, Cardiff, CF10 1DD', 'https://adamstastic.co.uk');
```

-- ContactDetails Table

```
INSERT INTO ContactDetails (ContactType, ContactValue)
VALUES
('Email', 'support@shauncycles.co.uk'),
('Phone', '+44 70770553'),
('Email', 'infoworld@garethbikes.co.uk'),
('Phone', '+44 12345678');
```

-- ManufacturerContactDetails Table

```
INSERT INTO ManufacturerContactDetails (ManufacturerID, ContactID)
VALUES
(1, 1),
(1, 2),
(2, 3),
```

Shaun James Davies (2317135)

```
(4, 4);
```

```
-- Category Table
```

```
INSERT INTO Category (CategoryName, CategoryDescription)
```

```
VALUES
```

```
('Road Bikes', 'High-performance bikes designed for road cycling.'),
```

```
('Mountain Bikes', 'Durable bikes built for off-road terrains.'),
```

```
('Hybrid Bikes', 'Versatile bikes suitable for both road and trails.'),
```

```
('Mystical Bikes', 'The bikes are the hidden gem of the store - magic bikes.');
```

```
-- Product Table
```

```
INSERT INTO Product (ProductName, ProductDescription, Price, StockQuantity, DiscountedPrice,  
LaunchDate, ManufacturerID, CategoryID)
```

```
VALUES
```

```
('Masterful 3000', 'A road bike.', 1299.99, 50, 1199.99, '2023-04-01', 1, 1),
```

```
('Fail Nay-ser', 'A mountain bike.', 999.99, 30, NULL, '2023-05-15', 2, 2),
```

```
('Scary ether', 'A hybrid bike.', 799.99, 40, 749.99, '2023-06-20', 3, 3),
```

```
('Magic Mark', 'A high-quality cycling helmet.', 59.99, 100, 49.99, '2023-07-01', 4, 4);
```

```
-- ProductCategory Table
```

```
INSERT INTO ProductCategory (ProductID, CategoryID)
```

```
VALUES
```

```
(1, 1),
```

```
(2, 2),
```

```
(3, 3),
```

```
(4, 4);
```

```
-- Tags Table
```

```
INSERT INTO Tags (TagName)
```

```
VALUES
```

```
('Lightweight'),
```

```
('Durable'),
```

```
('Eco-Friendly'),  
('Magical');
```

```
-- ProductTags Table
```

```
INSERT INTO ProductTags (ProductID, TagID)
```

```
VALUES
```

```
(1, 1),
```

```
(2, 2),
```

```
(3, 3),
```

```
(4, 4);
```

```
-- Dimensions Table
```

```
INSERT INTO Dimensions (ProductLength, ProductWidth, ProductHeight)
```

```
VALUES
```

```
(6.50, 9.50, 12.00),
```

```
(20.00, 7.70, 5.30),
```

```
(80.50, 100.50, 111.00),
```

```
(123.90, 120.65, 1201.25);
```

```
-- ProductDimensions Table
```

```
INSERT INTO ProductDimensions (ProductID, DimensionID)
```

```
VALUES
```

```
(1, 1),
```

```
(2, 2),
```

```
(3, 3),
```

```
(4, 4);
```

```
-- Customer Table
```

```
INSERT INTO Customer (CustomerName, PhoneNumber, CustomerPoints)
```

```
VALUES
```

```
('Dafydd Jones', '+44 123 456 7890', 50),
```

```
('Carys Williams', '+44 987 654 3210', 100),  
( 'Rhys Morgan', '+44 123 123 4567', 75),  
( 'Sian Evans', '+44 444 444 6543', 60);
```

-- OrderStatus Table

```
INSERT INTO OrderStatus (StatusName, OrderStatusDescription)  
VALUES  
( 'Pending', 'Waiting confirmation.'),  
( 'Confirmed', 'The Order has been confirmed.'),  
( 'Shipped', 'Shipped!!!.'),  
( 'Delivered', 'Delivered!!!!.');
```

-- Order Table

```
INSERT INTO `Order` (CustomerID, OrderDate, ShippingDate, OrderStatusID, TotalAmount)  
VALUES  
(1, '2023-08-15', '2024-06-21', 3, 1299.99),  
(2, '2023-08-16', '2024-01-01', 4, 999.99),  
(3, '2023-08-17', '2025-01-02', 2, 799.99),  
(4, '2023-08-18', '2025-01-03', 1, 59.99);
```

-- OrderDetails Table

```
INSERT INTO OrderDetails (OrderID, ProductID, Quantity, Price)  
VALUES  
(1, 1, 1, 1299.99),  
(2, 2, 1, 999.99),  
(3, 3, 1, 799.99),  
(4, 4, 1, 59.99);
```

-- Feedback Table

```
INSERT INTO Feedback (Rating, Comments, FeedbackDate, OverallExperience, ExperienceComments,  
ProductID, CustomerID)  
VALUES
```

Shaun James Davies (2317135)

```
(5, 'Cynnyrch rhagorol, cadarn iawn!', '2023-08-20', 5, 'Pryniant gorau erioed.', 1, 1),  
(4, 'Ansawdd da ond ychydig yn ddrud.', '2023-08-21', 4, 'Boddhaol iawn.', 2, 2),  
(3, 'Profiad cyffredin.', '2023-08-22', 3, 'Gallai fod yn well.', 3, 3),  
(5, 'Amazingly lightweight, the flight was the best part!', '2023-08-23', 5, 'Terrific :).', 4, 4);
```

-- Not Null Constraints test

-- Manufacturer Table

-- This test attempts to insert a row into manufacturer with the ManufacturerName NULL.

```
INSERT INTO Manufacturer (ManufacturerName, RegistrationDate, Address, Website)  
VALUES (NULL, '2023-09-01', '123 xxx, London, SW1C 1CC', 'https://xxx.co.uk');
```

-- This test, if the code is correct, should fail because ManufacturerName cannot be NULL.

-- Customer Table

-- This test tries to enter a customername as null which is against the constraint.

```
INSERT INTO Customer (CustomerName, PhoneNumber, CustomerPoints)  
VALUES (NULL, '+44 123 456 7890', 10);
```

-- -- This test, if the code is correct, should fail because CustomerName cannot be NULL.

-- Default Values

-- Customer Table

-- This test tries the scenario of entering no value for CustomerPoints.

```
INSERT INTO Customer (CustomerName, PhoneNumber)  
VALUES ('xxxx', '+44 777 123 4567');
```

-- -- This test, if the code is correct, should insert CustomerPoints as 0 by default.

-- Cascading Deletes

-- Delete Manufacturer and Check Cascade in Product Table

-- This test tries to delete Manufacturer with an ID of 1 to test if the update cascades through to product

DELETE FROM Manufacturer WHERE ManufacturerID = 1;

-- -- This test, if the code is correct, should delete all products linked to ManufacturerID 1 in the Product table.

-- Delete Product and Check Cascade in ProductCategory Table

-- This test is the same as above but for Product.

DELETE FROM Product WHERE ProductID = 1;

-- -- This test, if the code is correct, should delete the associated rows in ProductCategory table for ProductID 1.

-- Foreign Key Constraints

-- Product Table

-- This test tries to insert a product with a ManufacturerID 12345

INSERT INTO Product (ProductName, ProductDescription, Price, StockQuantity, DiscountedPrice, LaunchDate, ManufacturerID, CategoryID)

VALUES ('xxx', 'xxx', 100.00, 10, NULL, '2023-10-01', 12345, 1);

-- -- This test, if the code is correct, should fail because ManufacturerID 12345 does not exist in Manufacturer table.

-- ManufacturerContactDetails Table

-- This test uses ManufacturerID 12345 again to see if there's any contact details for this non existent Manufacturer

INSERT INTO ManufacturerContactDetails (ManufacturerID, ContactID)

VALUES (12345, 1);

-- -- This test, if the code is correct, should fail because ManufacturerID 12345 does not exist.

-- Primary Key and Auto-Increment Constraints

-- Manufacturer Table (Auto-Increment and Primary Key)

-- This test tries to make a ManufacturerID of 1 which already exists and shouldn't be allowed.

```
INSERT INTO Manufacturer (ManufacturerID, ManufacturerName, RegistrationDate, Address, Website)
```

```
VALUES (1, 'xxxx', '2019-01-01', '456 Shaun Street, London, SHA! 1BB', 'https://xxxx.co.uk');
```

```
-- -- This test, if the code is correct, should fail because ManufacturerID 1 already exists.
```

```
-- Check Constraints
```

```
-- Product Table (StockQuantity >= 0)
```

```
-- This test is trying to enter negative stock which shouldn't be allowed.
```

```
INSERT INTO Product (ProductName, ProductDescription, Price, StockQuantity, DiscountedPrice, LaunchDate, ManufacturerID, CategoryID)
```

```
VALUES ('xxxxxxxxx', 'xxxxxxx', 200.00, -5, NULL, '2019-10-01', 1, 1);
```

```
-- -- This test, if the code is correct, should fail because StockQuantity cannot be negative.
```

```
-- Feedback Table (Rating BETWEEN 1 AND 5)
```

```
-- This test is test that the rating constraint has been enforced
```

```
INSERT INTO Feedback (Rating, Comments, FeedbackDate, OverallExperience, ExperienceComments, ProductID, CustomerID)
```

```
VALUES (10, 'Test', '2024-12-25', 3, 'xxx.', 1, 1);
```

```
-- -- This test, if the code is correct, should fail because Rating is outside the range 1-5.
```

```
-- Composite Primary Key Constraints
```

```
-- ManufacturerContactDetails Table
```

```
-- This test is trying to add a ManufacturerID/ ContactID pairing that already exists which shouldn't be allowed
```

```
INSERT INTO ManufacturerContactDetails (ManufacturerID, ContactID)
```

```
VALUES (1, 1);
```

```
-- -- This test, if the code is correct, should fail because the combination of ManufacturerID and ContactID already exists.
```

```
-- Commit
```

```

START TRANSACTION; -- Begin the transaction in order to place an order

-- Insert a new order
INSERT INTO `Order` (CustomerID, OrderDate, ShippingDate, OrderStatusID, TotalAmount)
VALUES (1, '2023-11-24', '2023-12-24', 2, 1999.98);

-- Get the last inserted OrderID for setting up and connecting with OrderDetails
SET @OrderID = LAST_INSERT_ID();

-- Insert products into OrderDetails as per the diagram detailing relationship
INSERT INTO OrderDetails (OrderID, ProductID, Quantity, Price)
VALUES (@OrderID, 1, 1, 1299.99), -- Product 1
        (@OrderID, 2, 1, 699.99); -- Product 2

-- Update stock quantities for purchased products; this allows for accurate measurement of stock
UPDATE Product
SET StockQuantity = StockQuantity - 1
WHERE ProductID IN (1, 2);

-- Update customer points
UPDATE Customer
SET CustomerPoints = CustomerPoints + 20
WHERE CustomerID = 1;

-- Commit the transaction to finalise all changes
COMMIT;

DELIMITER //

CREATE PROCEDURE PlaceOrderWithStockCheck(
    IN customer_id INT,

```

```

    IN product_id INT,
    IN quantity INT,
    IN order_date DATE,
    IN shipping_date DATE,
    IN order_status INT,
    IN total_amount DECIMAL(10,2)
)
BEGIN
    DECLARE stock_available INT;

    -- Start the transaction
    START TRANSACTION;

    -- Check stock availability
    SELECT StockQuantity INTO stock_available
    FROM Product
    WHERE ProductID = product_id;

    IF stock_available < quantity THEN
        -- Insufficient stock, rollback transaction
        ROLLBACK;
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Insufficient stock for the requested product.';
    ELSE
        -- Insert order
        INSERT INTO `Order` (CustomerID, OrderDate, ShippingDate, OrderStatusID, TotalAmount)
        VALUES (customer_id, order_date, shipping_date, order_status, total_amount);

        -- Get the last inserted OrderID for linking with OrderDetails
        SET @OrderID = LAST_INSERT_ID();
    
```

```

-- Insert order details

INSERT INTO OrderDetails (OrderID, ProductID, Quantity, Price)
VALUES (@OrderID, product_id, quantity, total_amount / quantity);

-- Update stock quantities

UPDATE Product
SET StockQuantity = StockQuantity - quantity
WHERE ProductID = product_id;

-- Commit the transaction

COMMIT;

END IF;

END //

DELIMITER ;

-- Example Execution:

CALL PlaceOrderWithStockCheck(2, 3, 2, '2024-12-02', '2024-12-06', 1, 500.00);

-- Indexes for Product Searches

CREATE INDEX idx_ProductName ON Product (ProductName);

-- Speeds up searches by product name select for index. The product name is the main attribute across
the platform

CREATE INDEX idx_CategoryID ON Product (CategoryID);

-- Optimises filtering by category as users will likely use this option regularly

-- Indexes for Order Lookups

CREATE INDEX idx_CustomerID ON `Order` (CustomerID);

-- looking up by customer seems to be a useful tool in any business, this will be useful

CREATE INDEX idx_OrderDate ON `Order` (OrderDate);

-- This will be useful in order to filter efficiently on a popular attribute; OrderDate

```