



ADVENTUREWORKS CYCLES' DATABASE DESIGN & IMPLEMENTATION

Databases and Application Development
(ACCA5026)

Joe Andrew Frohwein (2315545)
Computing (Software Engineering)

Contents

Database Design.....	2
Introduction.....	2
Entity Relationship Diagram	3
Database Implementation	4
Table Structure.....	4
Referential Integrity Constraints	6
Check Constraints	6
Table Population	7
Constraints Testing.....	11
Transactions.....	13
Performance Tuning Enhancement	14
References	20
Appendix	21

Database Design

Introduction

Specialising in a diverse range of road bikes, mountain bikes and hybrid bikes, as well as various cycling gear and accessories, are AdventureWorks Cycles, a prominent bicycle manufacturer in the heart of the city. Whilst being a leader in the cycling industry for decades, the company has developed a strong reputation within its community.

In an effort to modernize their operations, expand their online reach and presence, and continue to maintain their status in the market, the company intends to create an advanced online marketplace. This platform will allow cycling enthusiasts to explore, purchase, and customize bicycles and accessories from renowned manufacturers worldwide. In order to achieve this, the company recognises the need for a robust data infrastructure to support their planned platform, allowing for large amounts of data management.

Therefore, AdventureWorks Cycles has requested consultation on how to identify and implement the most suitable database solution for their online marketplace.

During the initial consultation, it was determined that the database will need to store large amounts of data on the company's products, manufacturers, product categories, customers and orders.

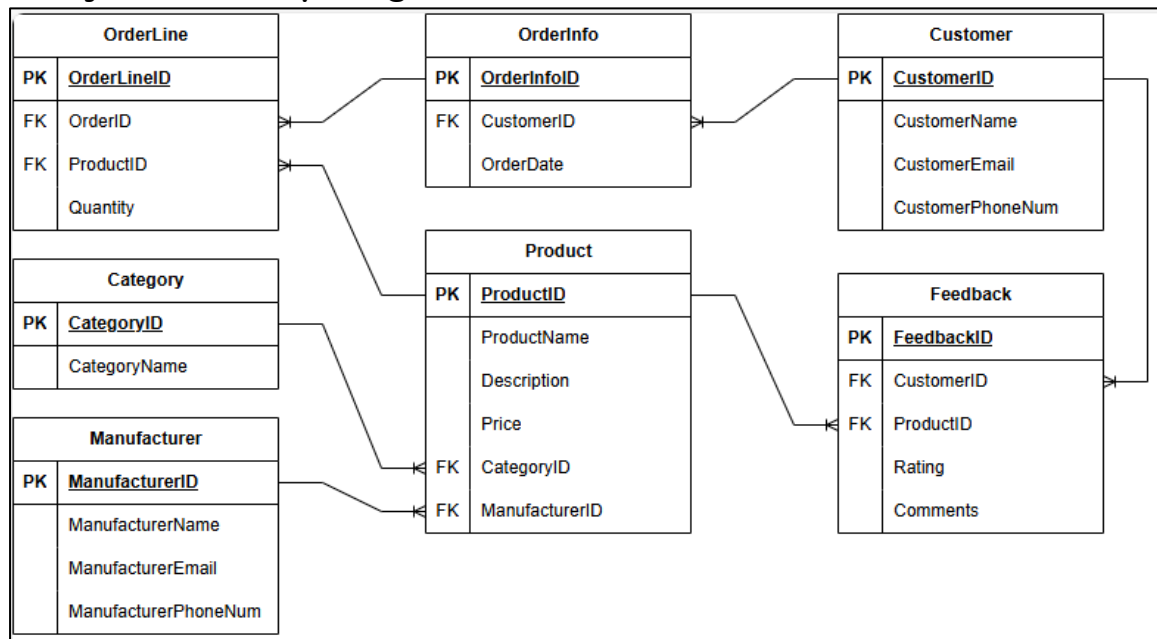
It was also determined that the platform will need to be able to perform various processing tasks. These tasks will include:

- Manufacturer registration and product listing, allowing manufacturers to provide their own contact details, as well as list their products up onto the website.
- Customer registration, allowing the customer to create an account.
- Product purchasing, allowing the customers to browse, place orders and purchase products through their accounts.
- Order processing, allows the system to process orders and calculate totals and stock values etc.
- Feedback, allowing the customers to leave reviews on the products which they've purchased.

Overall, the primary requirements include efficient management of user and manufacturer data, product and category information, order processing, and feedback management.

Additionally, as the company has many staff, the database must support concurrent user access. Therefore there must be suitable stored procedures for insert, update, and delete operations, including rollback and commit functionalities.

Entity Relationship Diagram



The image above is the Entity Relationship Diagram which represents Adventure Works Cycles' database schema.

The ERD shows each table's name, column names, primary keys and foreign keys, as well as the relationships between primary and foreign keys.

The schema consists of tables representing the main entities that were outlined in the brief. These entities are:

- **Manufacturer** – Manufacturer details
- **Category** – Categories which products fall under
- **Product** – Product details
- **Customer** – Customer details
- **OrderInfo** – Overall details of orders placed by customers
- **OrderLine** – Each individual product ordered on each overall order
- **Feedback** – Customer reviews

These tables have been designed so that they're normalised to 3rd Normal Form (3NF). For a table to be in 3NF, the following criteria must be achieved:

- **1st Normal Form (1NF)**
 - All column must be atomic
 - Each row must be unique
- **2nd Normal Form (2NF)**
 - 1NF is achieved
 - No partial dependency
- **3rd Normal Form (3NF)**
 - 2NF is achieved
 - No transitive dependency

As there are no repeating groups, all non-primary columns are fully functionally dependent on the primary, and all non-primary key attributes are defined using another non-primary key attribute, the database schema achieves 3NF. [1]

Database Implementation

Table Structure

The screenshots in this section show the process of creating the database and the tables, which were designed in the Entity Relationship Diagram in the previous section. A screenshot of each created table is also shown.

Database creation

```
-- Creates and assigns the database as active
CREATE DATABASE assignment;
USE assignment;
```

Manufacturer table

```
-- Creates the Manufacturer table to store Manufacturer's names and contact details
CREATE TABLE Manufacturer (
    ManufacturerID INT AUTO_INCREMENT PRIMARY KEY,
    ManufacturerName CHAR(100) NOT NULL,
    ManufacturerEmail CHAR(100),
    ManufacturerPhoneNum CHAR(15)
);
```

	ManufacturerID	ManufacturerName	ManufacturerEmail	ManufacturerPhoneNum
▶*	NULL	NULL	NULL	NULL

Category table

```
-- Creates the Category table to store product type classification information
CREATE TABLE Category (
    CategoryID INT AUTO_INCREMENT PRIMARY KEY,
    CategoryName CHAR(100) NOT NULL
);
```

	CategoryID	CategoryName
*	NULL	NULL

Product table

```
-- Creates the Product table to store the information on the items available for sale
-- There can be many products from one category
-- There can be many products sold by one manufacturer
CREATE TABLE Product (
    ProductID INT AUTO_INCREMENT PRIMARY KEY,
    ProductName CHAR(100) NOT NULL,
    ProductDescription CHAR(200),
    Price DECIMAL(10,2),
    CategoryID INT,
    ManufacturerID INT
);
```

	ProductID	ProductName	ProductDescription	Price	CategoryID	ManufacturerID
*	NULL	NULL	NULL	NULL	NULL	NULL

Customer table

```
-- Creates the Customer table to store the information of the individuals or organizations that purchase products
CREATE TABLE Customer (
  CustomerID INT AUTO_INCREMENT PRIMARY KEY,
  CustomerName CHAR(100) NOT NULL,
  CustomerEmail CHAR(100),
  CustomerPhoneNum CHAR(15)
);
```

	CustomerID	CustomerName	CustomerEmail	CustomerPhoneNum
*	NULL	NULL	NULL	NULL

OrderInfo table

```
-- Creates the OrderInfo table to store the overall information of each order
-- There can be many orders placed by one customer
CREATE TABLE OrderInfo (
  OrderInfoID INT AUTO_INCREMENT PRIMARY KEY,
  CustomerID INT,
  OrderDate DATETIME
);
```

	OrderInfoID	CustomerID	OrderDate
*	NULL	NULL	NULL

OrderLine table

```
-- Creates the OrderLine table to store each individual product purchase and what order it belongs to
-- There can be many OrderLines assigned to one OrderInfo
CREATE TABLE OrderLine (
  OrderLineID INT AUTO_INCREMENT PRIMARY KEY,
  OrderInfoID INT,
  ProductID INT,
  Quantity INT NOT NULL
);
```

	OrderLineID	OrderInfoID	ProductID	Quantity
*	NULL	NULL	NULL	NULL

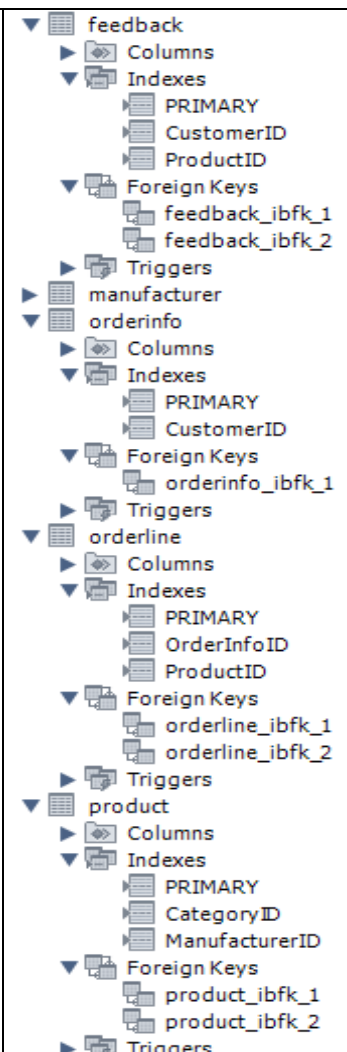
Feedback table

```
-- Creates the Feedback table to store feedback and ratings provided by the customers
-- There can be many sets of feedback given by one customer
-- There can be many sets of feedback given for one product
CREATE TABLE Feedback (
  FeedbackID INT AUTO_INCREMENT PRIMARY KEY,
  CustomerID INT,
  ProductID INT,
  Rating INT NOT NULL,
  Comments CHAR(250)
);
```

	FeedbackID	CustomerID	ProductID	Rating	Comments
*	NULL	NULL	NULL	NULL	NULL

Referential Integrity Constraints

The screenshots in this section show the process of implementing the referential integrity checks, as well as the created indexes and foreign keys within the database schema.

<pre>-- Assigns CategoryID and ManufacturerID as Foreign keys in the Product table ALTER TABLE Product ADD FOREIGN KEY (CategoryID) REFERENCES Category(CategoryID), ADD FOREIGN KEY (ManufacturerID) REFERENCES Manufacturer(ManufacturerID); -- Assigns CustomerID as Foreign keys in the OrderInfo table ALTER TABLE OrderInfo ADD FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID); -- Assigns OrderInfoID and ProductID as Foreign keys in the OrderLine table ALTER TABLE OrderLine ADD FOREIGN KEY (OrderInfoID) REFERENCES OrderInfo(OrderInfoID), ADD FOREIGN KEY (ProductID) REFERENCES Product(ProductID); -- Assigns CustomerID and ProductID as Foreign keys in the Feedback table ALTER TABLE Feedback ADD FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID), ADD FOREIGN KEY (ProductID) REFERENCES Product(ProductID);</pre>	
--	--

Check Constraints

The screenshot in this section shows the process of implementing the check constraints, which are tested and documented in the 'Constraints Testing' section below.

<pre>-- Assigns a check constraint to the Product table, ensuring that Price is a positive value as a price cannot be negative ALTER TABLE Product ADD CONSTRAINT checkPrice CHECK (Price > 0); -- Assigns a check constraint to the OrderLine table, ensuring that Quantity a positive value as a quantity can not be negative ALTER TABLE OrderLine ADD CONSTRAINT checkQuantity CHECK (Quantity > 0); -- Assigns a check constraint to the Feedback table, ensuring that Rating is a value in the range 1-5 inclusive, to act as a 5 star rating ALTER TABLE Feedback ADD CONSTRAINT checkRating CHECK (Rating BETWEEN 1 AND 5);</pre>

Table Population

The screenshots in this section show the process of populating the previously created tables with realistic sample data. A screenshot of each populated table is also shown.

Manufacturer table

```
-- Inserts sample manufacturer records
INSERT INTO Manufacturer (ManufacturerName, ManufacturerEmail, ManufacturerPhoneNum)
VALUES
('British Bikes Ltd', 'contact@britishbikes.co.uk', '02079461234'),
('CycleWorks UK', 'sales@cycleworksuk.com', '01923847561'),
('Swansea Cycles', 'info@swanseacycles.co.uk', '01792476543'),
('Welsh Bicycles', 'support@welshbicycles.co.uk', '02920458932'),
('PedalPower Ltd', 'contact@pedalpower.co.uk', '02031465987');
```

	ManufacturerID	ManufacturerName	ManufacturerEmail	ManufacturerPhoneNum
▶	1	British Bikes Ltd	contact@britishbikes.co.uk	02079461234
	2	CycleWorks UK	sales@cycleworksuk.com	01923847561
	3	Swansea Cycles	info@swanseacycles.co.uk	01792476543
	4	Welsh Bicycles	support@welshbicycles.co.uk	02920458932
	5	PedalPower Ltd	contact@pedalpower.co.uk	02031465987
⚙	NULL	NULL	NULL	NULL

Category table

```
-- Inserts sample category records
INSERT INTO Category (CategoryName)
VALUES
('Road Bikes'),
('Mountain Bikes'),
('Hybrid Bikes'),
('Cycling Accessories');
```

	CategoryID	CategoryName
▶	1	Road Bikes
	2	Mountain Bikes
	3	Hybrid Bikes
	4	Cycling Accessories
⚙	NULL	NULL

Product table

```
-- Inserts sample product records
INSERT INTO Product (ProductName, ProductDescription, Price, CategoryID, ManufacturerID)
VALUES
('Road Racer 500', 'Lightweight, fast road bike for professional cyclists', 999.99, 1, 1),
('Mountain Explorer X', 'Durable mountain bike for tough terrains', 799.99, 2, 2),
('Hybrid Trail 700', 'Versatile hybrid bike for both road and light trails', 650.00, 3, 3),
('Cycling Gloves', 'Comfortable and durable cycling gloves', 29.99, 4, 4),
('Mountain Helmet Pro', 'High-quality helmet for mountain biking', 49.99, 4, 2),
('Speedster 9000', 'Top-end racing bike for long-distance speed', 1199.99, 1, 5),
('City Cruiser 300', 'Comfortable bike for city commuting', 450.00, 3, 3),
('Bike Repair Kit', 'Comprehensive bike maintenance and repair kit', 59.99, 4, 5),
('Trail Blazer XT', 'Heavy-duty mountain bike for extreme trails', 899.99, 2, 1),
('Road Racer 700', 'Advanced road bike designed for high performance', 1300.00, 1, 4),
('Speedster Pro', 'Professional racing bike for speed enthusiasts', 1500.00, 1, 1),
('Urban Cruiser', 'Stylish and reliable city bike for everyday commuting', 550.00, 3, 2);
```

	ProductID	ProductName	ProductDescription	Price	CategoryID	ManufacturerID
▶	1	Road Racer 500	Lightweight, fast road bike for professional cycli...	999.99	1	1
	2	Mountain Explorer X	Durable mountain bike for tough terrains	799.99	2	2
	3	Hybrid Trail 700	Versatile hybrid bike for both road and light trails	650.00	3	3
	4	Cycling Gloves	Comfortable and durable cycling gloves	29.99	4	4
	5	Mountain Helmet Pro	High-quality helmet for mountain biking	49.99	4	2
	6	Speedster 9000	Top-end racing bike for long-distance speed	1199.99	1	5
	7	City Cruiser 300	Comfortable bike for city commuting	450.00	3	3
	8	Bike Repair Kit	Comprehensive bike maintenance and repair kit	59.99	4	5
	9	Trail Blazer XT	Heavy-duty mountain bike for extreme trails	899.99	2	1
	10	Road Racer 700	Advanced road bike designed for high performa...	1300.00	1	4
	11	Speedster Pro	Professional racing bike for speed enthusiasts	1500.00	1	1
	12	Urban Cruiser	Stylish and reliable city bike for everyday comm...	550.00	3	2
*	NULL	NULL	NULL	NULL	NULL	NULL

Customer table

```
-- Inserts sample customer records
INSERT INTO Customer (CustomerName, CustomerEmail, CustomerPhoneNum)
VALUES
('John Doe', 'john.doe@example.com', '07812345678'),
('Jane Smith', 'jane.smith@example.com', '07898765432'),
('Mike Johnson', 'mike.johnson@example.com', '07965432109'),
('Emily Davis', 'emily.davis@example.com', '07788990022'),
('Paul Brown', 'paul.brown@example.com', '07722334455');
```

	CustomerID	CustomerName	CustomerEmail	CustomerPhoneNum
▶	1	John Doe	john.doe@example.com	07812345678
	2	Jane Smith	jane.smith@example.com	07898765432
	3	Mike Johnson	mike.johnson@example.com	07965432109
	4	Emily Davis	emily.davis@example.com	07788990022
	5	Paul Brown	paul.brown@example.com	07722334455
*	NULL	NULL	NULL	NULL

OrderInfo table

```
-- Inserts sample order records
INSERT INTO OrderInfo (CustomerID, OrderDate)
VALUES
(3, '2024-12-01 10:30:00'), -- Mike Johnson, 1st order
(5, '2024-12-02 14:45:00'), -- Paul Brown, 1st order
(1, '2024-12-03 13:15:00'), -- John Doe, 1st order
(4, '2024-12-04 09:30:00'), -- Emily Davis, 1st order
(2, '2024-12-05 16:25:00'), -- Jane Smith, 1st order
(5, '2024-12-06 11:55:00'), -- Paul Brown, 2nd order
(3, '2024-12-07 17:30:00'), -- Mike Johnson, 2nd order
(1, '2024-12-08 13:05:00'), -- John Doe, 2nd order
(4, '2024-12-09 15:45:00'); -- Emily Davis, 2nd order
```

	OrderInfoID	CustomerID	OrderDate
▶	1	3	2024-12-01 10:30:00
	2	5	2024-12-02 14:45:00
	3	1	2024-12-03 13:15:00
	4	4	2024-12-04 09:30:00
	5	2	2024-12-05 16:25:00
	6	5	2024-12-06 11:55:00
	7	3	2024-12-07 17:30:00
	8	1	2024-12-08 13:05:00
	9	4	2024-12-09 15:45:00
*	NULL	NULL	NULL

OrderLine table

<pre>-- Inserts sample order line records INSERT INTO OrderLine (OrderInfoID, ProductID, Quantity) VALUES (1, 1, 2), -- OrderInfoID 1, ProductID 1 (Road Racer 500), 2 units (1, 3, 1), -- OrderInfoID 1, ProductID 3 (Hybrid Trail 700), 1 unit (2, 2, 3), -- OrderInfoID 2, ProductID 2 (Mountain Explorer X), 3 units (2, 4, 2), -- OrderInfoID 2, ProductID 4 (Cycling gloves), 2 units (3, 3, 1), -- OrderInfoID 3, ProductID 3 (Hybrid Trail 700), 1 unit (3, 1, 1), -- OrderInfoID 3, ProductID 1 (Road Racer 500), 1 unit (4, 5, 1), -- OrderInfoID 4, ProductID 5 (Mountain Helmet Pro), 1 unit (4, 6, 2), -- OrderInfoID 4, ProductID 6 (Speedster 9000), 2 units (5, 7, 1), -- OrderInfoID 5, ProductID 7 (City Cruiser 300), 1 unit (5, 9, 1), -- OrderInfoID 5, ProductID 9 (Bike Repair Kit), 1 unit (6, 8, 2), -- OrderInfoID 6, ProductID 8 (Trail Blazer XT), 2 units (6, 1, 1), -- OrderInfoID 6, ProductID 1 (Road Racer 500), 1 unit (7, 3, 1), -- OrderInfoID 7, ProductID 3 (Hybrid Trail 700), 1 unit (7, 4, 1), -- OrderInfoID 7, ProductID 4 (Cycling Gloves), 1 unit (7, 6, 2), -- OrderInfoID 7, ProductID 6 (Speedster 9000), 2 units (8, 2, 2), -- OrderInfoID 8, ProductID 2 (Mountain Explorer X), 2 units (9, 5, 1), -- OrderInfoID 9, ProductID 5 (Mountain Helmet Pro), 1 unit (9, 9, 1); -- OrderInfoID 9, ProductID 9 (Bike Repair Kit), 1 unit</pre>				
	OrderLineID	OrderInfoID	ProductID	Quantity
▶	1	1	1	2
	2	1	3	1
	3	2	2	3
	4	2	4	2
	5	3	3	1
	6	3	1	1
	7	4	5	1
	8	4	6	2
	9	5	7	1
	10	5	9	1
	11	6	8	2
	12	6	1	1
	13	7	3	1
	14	7	4	1
	15	7	6	2
	16	8	2	2
	17	9	5	1
	18	9	9	1
*	NULL	NULL	NULL	NULL

Feedback table

```
-- Inserts sample feedback records
INSERT INTO Feedback (CustomerID, ProductID, Rating, Comments)
VALUES
(3, 3, 4, 'Hybrid Trail 700 is a great bike for both roads and light trails.'),
(3, 4, 3, 'Cycling gloves are good, but not the best for longer rides.'),
(1, 1, 5, 'The Road Racer 500 is top-notch. Lightweight and performs excellently.'),
(1, 3, 4, 'Hybrid Trail 700 is a solid choice for all-around cycling. Good value for money.'),
(1, 2, 5, 'The Mountain Explorer X is perfect for off-road biking. Very sturdy!'),
(4, 5, 5, 'The Mountain Helmet Pro gives great protection, very comfortable to wear.'),
(4, 6, 5, 'The Speedster 9000 is perfect for racing and long rides. Highly recommend!'),
(5, 8, 4, 'Trail Blazer XT is great for tough mountain trails, but a bit heavy.'),
(5, 2, 5, 'The Mountain Explorer X is perfect for off-road biking. Very sturdy!'),
(5, 4, 4, 'Cycling gloves are comfortable and durable, would recommend.'),
(2, 7, 3, 'The City Cruiser 300 is decent for city commuting, but could be more comfortable.'),
(2, 9, 5, 'The Bike Repair Kit has everything I need to keep my bike in top condition.');
```

	FeedbackID	CustomerID	ProductID	Rating	Comments
▶	1	3	3	4	Hybrid Trail 700 is a great bike for both roads and light trails.
	2	3	4	3	Cycling gloves are good, but not the best for longer rides.
	3	1	1	5	The Road Racer 500 is top-notch. Lightweight and performs excellently.
	4	1	3	4	Hybrid Trail 700 is a solid choice for all-around cycling. Good value for money.
	5	1	2	5	The Mountain Explorer X is perfect for off-road biking. Very sturdy!
	6	4	5	5	The Mountain Helmet Pro gives great protection, very comfortable to wear.
	7	4	6	5	The Speedster 9000 is perfect for racing and long rides. Highly recommend!
	8	5	8	4	Trail Blazer XT is great for tough mountain trails, but a bit heavy.
	9	5	2	5	The Mountain Explorer X is perfect for off-road biking. Very sturdy!
	10	5	4	4	Cycling gloves are comfortable and durable, would recommend.
	11	2	7	3	The City Cruiser 300 is decent for city commuting, but could be more comfortable.
	12	2	9	5	The Bike Repair Kit has everything I need to keep my bike in top condition.
✱	NULL	NULL	NULL	NULL	NULL

Constraints Testing

The screenshots in this section show the process of testing the functionality of the check constraints. Each check is tested with invalid data to demonstrate the functionality. Each table is shown after to demonstrate that the invalid data has not been accepted and added to the table, showing that the constraint is working correctly. Finally, the results of the testing are shown to show that each test returned an error message.

Products table constraints

```
-- Inserts new records into Products table
-- The records are invalid as it does not pass the 'checkPrice CHECK (Price > 0)' constraint
-- The records will therefore be rejected
INSERT INTO Product (ProductName, ProductDescription, Price, CategoryID, ManufacturerID)
VALUES
('Road Racer 1000', 'The upgraded version of the Road Racer 500. Lighter, faster, better!', -999.99, 1, 1);

INSERT INTO Product (ProductName, ProductDescription, Price, CategoryID, ManufacturerID)
VALUES
('A bike', 'Just a bike.', 0.00, 1, 1);
```

	ProductID	ProductName	ProductDescription	Price	CategoryID	ManufacturerID
▶	1	Road Racer 500	Lightweight, fast road bike for professional cycli...	999.99	1	1
	2	Mountain Explorer X	Durable mountain bike for tough terrains	799.99	2	2
	3	Hybrid Trail 700	Versatile hybrid bike for both road and light trails	650.00	3	3
	4	Cycling Gloves	Comfortable and durable cycling gloves	29.99	4	4
	5	Mountain Helmet Pro	High-quality helmet for mountain biking	49.99	4	2
	6	Speedster 9000	Top-end racing bike for long-distance speed	1199.99	1	5
	7	City Cruiser 300	Comfortable bike for city commuting	450.00	3	3
	8	Bike Repair Kit	Comprehensive bike maintenance and repair kit	59.99	4	5
	9	Trail Blazer XT	Heavy-duty mountain bike for extreme trails	899.99	2	1
	10	Road Racer 700	Advanced road bike designed for high performa...	1300.00	1	4
	11	Speedster Pro	Professional racing bike for speed enthusiasts	1500.00	1	1
	12	Urban Cruiser	Stylish and reliable city bike for everyday comm...	550.00	3	2
✱	NULL	NULL	NULL	NULL	NULL	NULL

OrderLine table constraints

```
-- Inserts new records into OrderLine table
-- The records are invalid as they do not pass the 'checkQuantity CHECK (Quantity > 0)' constraint
-- The records will therefore be rejected
INSERT INTO OrderLine (OrderInfoID, ProductID, Quantity)
VALUES
(9, 1, 0);

INSERT INTO OrderLine (OrderInfoID, ProductID, Quantity)
VALUES
(9, 2, -1);
```

	OrderLineID	OrderInfoID	ProductID	Quantity
▶	1	1	1	2
	2	1	3	1
	3	2	2	3
	4	2	4	2
	5	3	3	1
	6	3	1	1
	7	4	5	1
	8	4	6	2
	9	5	7	1
	10	5	9	1
	11	6	8	2
	12	6	1	1
	13	7	3	1
	14	7	4	1
	15	7	6	2
	16	8	2	2
	17	9	5	1
	18	9	9	1
✱	NULL	NULL	NULL	NULL

Feedback table constraints

```
-- Inserts new records into Feedback table
-- The records are invalid as they do not pass the 'checkRating CHECK (Rating BETWEEN 1 AND 5)' constraint
-- The records will therefore be rejected
INSERT INTO Feedback (CustomerID, ProductID, Rating, Comments)
VALUES
(3, 6, 6, 'An amazing bike, my favourite one yet!');

INSERT INTO Feedback (CustomerID, ProductID, Rating, Comments)
VALUES
(4, 9, 0, 'The Trail Blazer XT is an awful bike, do not recommend at all!');
```

	FeedbackID	CustomerID	ProductID	Rating	Comments
▶	1	3	3	4	Hybrid Trail 700 is a great bike for both roads and light trails.
	2	3	4	3	Cycling gloves are good, but not the best for longer rides.
	3	1	1	5	The Road Racer 500 is top-notch. Lightweight and performs excellently.
	4	1	3	4	Hybrid Trail 700 is a solid choice for all-around cycling. Good value for money.
	5	1	2	5	The Mountain Explorer X is perfect for off-road biking. Very sturdy!
	6	4	5	5	The Mountain Helmet Pro gives great protection, very comfortable to wear.
	7	4	6	5	The Speedster 9000 is perfect for racing and long rides. Highly recommend!
	8	5	8	4	Trail Blazer XT is great for tough mountain trails, but a bit heavy.
	9	5	2	5	The Mountain Explorer X is perfect for off-road biking. Very sturdy!
	10	5	4	4	Cycling gloves are comfortable and durable, would recommend.
	11	2	7	3	The City Cruiser 300 is decent for city commuting, but could be more comfortable.
	12	2	9	5	The Bike Repair Kit has everything I need to keep my bike in top condition.
✱	NULL	NULL	NULL	NULL	NULL

Constraint testing results

#	Time	Action	Message
1	15:06:56	INSERT INTO Product (ProductName, ProductDescription, Price, CategoryID, ManufacturerID) VALUES ('Road Racer 1000', 'The upgraded version of the Road Racer 500. Lighter, faster, better!', 999.99, 1, 1)	Error Code: 3819. Check constraint 'checkPrice' is violated.
2	15:07:17	INSERT INTO Product (ProductName, ProductDescription, Price, CategoryID, ManufacturerID) VALUES ('Abike', 'Just a bike.', 0.00, 1, 1)	Error Code: 3819. Check constraint 'checkPrice' is violated.
3	15:07:44	INSERT INTO OrderLine (OrderInfoID, ProductID, Quantity) VALUES (9, 1, 0)	Error Code: 3819. Check constraint 'checkQuantity' is violated.
4	15:07:56	INSERT INTO OrderLine (OrderInfoID, ProductID, Quantity) VALUES (9, 2, -1)	Error Code: 3819. Check constraint 'checkQuantity' is violated.
5	15:08:10	INSERT INTO Feedback (CustomerID, ProductID, Rating, Comments) VALUES (3, 6, 6, 'An amazing bike, my favourite one yet!')	Error Code: 3819. Check constraint 'checkRating' is violated.
6	15:08:22	INSERT INTO Feedback (CustomerID, ProductID, Rating, Comments) VALUES (4, 9, 0, 'The Trail Blazer XT is an awful bike, do not recommend at all!')	Error Code: 3819. Check constraint 'checkRating' is violated.

Transactions

A database transaction is “a logical unit that generally includes several low-level steps. If one step of the transaction fails, the whole transaction fails. A database transaction is used to create, update, or retrieve data” [2]. In the case of Adventure Works Cycles, transactions could be used to control in the situation of a refund/exchange. The following example shows a customer purchasing a product, and then returning and swapping that product for an alternative. The following screenshot shows the transactional scenario which demonstrates the use of rollback and commit operations [3], [4].

The transaction is started and a new OrderLine record is added, applying to an already existing OrderInfo record, therefore adding the new product (ProductID = 1) to their existing order. This change is committed to ensure that this rollback is saved. However, the customer decides to return this item, so the process is rolled back to remove the previously inserted OrderLine record. The customer then decides to add 2 new products (ProductID = 2 and 4) to their order. As this is the customer’s final change to the order, this change is committed.

The OrderInfo and OrderLine tables are displayed below to demonstrate that the only changes made are the final INSERT statements, adding products 2 and 4 to OrderInfoID = 9 within the OrderLine table [5].

```
-- Assigns the database as active
USE assignment;

-- Begins the first transaction
START TRANSACTION;

-- Inserts into the OrderLine table 1 record for OrderInfoID = 9, as Emily Davis decided to add a Road Racer 500 onto her order
INSERT INTO OrderLine (OrderInfoID, ProductID, Quantity)
VALUES
(9, 1, 1); -- OrderInfoID 9, ProductID 1 (Road Racer 500), 1 unit

-- However, Emily Davis decided to return her Road Racer 500 and exchange the bike for a different bike
-- Rollback the second transaction due to refund/exchange
ROLLBACK;

-- Commits the rollback to close the first transaction
COMMIT;

-- Begins the second transaction
START TRANSACTION;

-- Inserts into the OrderLine table 2 records for OrderInfoID 9, as Emily Davis has exchanged her Road Racer 500 bike for a Mountain Explorer X and Cycling Gloves
INSERT INTO OrderLine (OrderInfoID, ProductID, Quantity)
VALUES
(9, 2, 1), -- OrderInfoID 10, ProductID 1 (Mountain Explorer X), 1 unit
(9, 4, 1); -- OrderInfoID 10, ProductID 4 (Cycling Gloves), 1 unit

-- Commits the changes made to the OrderLine table in the third transaction
COMMIT;

SELECT * FROM OrderInfo;
SELECT * FROM OrderLine;
```

	OrderInfoID	CustomerID	OrderDate
▶	1	3	2024-12-01 10:30:00
	2	5	2024-12-02 14:45:00
	3	1	2024-12-03 13:15:00
	4	4	2024-12-04 09:30:00
	5	2	2024-12-05 16:25:00
	6	5	2024-12-06 11:55:00
	7	3	2024-12-07 17:30:00
	8	1	2024-12-08 13:05:00
	9	4	2024-12-09 15:45:00
✱	NULL	NULL	NULL

OrderLineID	OrderInfoID	ProductID	Quantity
6	3	1	1
7	4	5	1
8	4	6	2
9	5	7	1
10	5	9	1
11	6	8	2
12	6	1	1
13	7	3	1
14	7	4	1
15	7	6	2
16	8	2	2
17	9	5	1
18	9	9	1
20	9	2	1
21	9	4	1
✱	NULL	NULL	NULL

Performance Tuning Enhancement

A database query is “a request for data from a database” and is “an essential component that enables the smooth functioning of databases, allowing users to retrieve, manipulate, and analyse data efficiently.” [6]. In the case of Adventure Works Cycles, queries can be used to retrieve important information and perform frequently executed procedures within the database. As well as this, the database’s performance can be further fine tuned by implementing secondary indexes. A secondary index is “a way to efficiently access records in a database (the primary) by means of some piece of information other than the usual (primary) key” [7] meaning that it allows non-primary keys (which are frequently used to search for/by) to be indexed, allowing for faster and more efficient data retrieval. Therefore, each column that has been implemented as a secondary key has been selected due to it’s presence within the queries, typically being searched for within the WHERE clause of the query.

Manufacturer table

As ManufacturerName is used within the following queries, a secondary index has been implemented to it to improve the performance and retrieval speeds.

When searching for Manufacturer details, data retrieval could be sped up using queries to filter by ID or name, which are both demonstrated and shown below.

Manufacturer information must be easily and quickly accessible as it may be required to provide product support to a customer, so to avoid keeping the customer waiting.

```
-- Manufacturer table -----  
-- Secondary indexes  
CREATE INDEX idx_ManufacturerName ON Manufacturer(ManufacturerName);  
  
-- A query to view a Manufacturer when given a ManufacturerID  
SELECT *  
FROM Manufacturer  
WHERE ManufacturerID = 2  
LIMIT 1;  
  
-- A query to view a Manufacturer when given a ManufacturerName  
SELECT *  
FROM Manufacturer  
WHERE ManufacturerName = 'Welsh Bicycles'  
LIMIT 1;
```

	ManufacturerID	ManufacturerName	ManufacturerEmail	ManufacturerPhoneNum
▶	2	CycleWorks UK	sales@cycleworksuk.com	01923847561
*	NULL	NULL	NULL	NULL

	ManufacturerID	ManufacturerName	ManufacturerEmail	ManufacturerPhoneNum
▶	4	Welsh Bicycles	support@welshbicycles.co.uk	02920458932
*	NULL	NULL	NULL	NULL

Product table

As CategoryID, ManufacturerID and Price are used within the following queries, secondary indexes have been implemented to them to improve the performance and retrieval speeds.

The 3 following queries are used to quickly display specific product information. The first and second queries are used to display a specific product, based on its ID or name. The third query is used to display any products that have a price which falls between a specified price range.

```
-- Product table -----
-- Secondary indexes
CREATE INDEX idx_CategoryID ON Product(CategoryID);
CREATE INDEX idx_ManufacturerID ON Product(ManufacturerID);
CREATE INDEX idx_Price ON Product(Price);

-- A query to view a Product when given a ProductID
SELECT ProductID, ProductName, Price, ManufacturerID, CategoryID
FROM Product
WHERE CategoryID = 3
ORDER BY ProductName ASC;

-- A query to view a Product when given a ProductName
SELECT ProductID, ProductName, Price, ManufacturerID, CategoryID
FROM Product
WHERE ManufacturerID = 2
ORDER BY ProductName ASC;

-- A query to view a Product when given a Price range
SELECT ProductID, ProductName, Price, ManufacturerID, CategoryID
FROM Product
WHERE Price BETWEEN 0 AND 500
ORDER BY Price ASC;
```

	ProductID	ProductName	Price	ManufacturerID	CategoryID
▶	7	City Cruiser 300	450.00	3	3
	3	Hybrid Trail 700	650.00	3	3
	12	Urban Cruiser	550.00	2	3
*	NULL	NULL	NULL	NULL	NULL

	ProductID	ProductName	Price	ManufacturerID	CategoryID
▶	2	Mountain Explorer X	799.99	2	2
	5	Mountain Helmet Pro	49.99	2	4
	12	Urban Cruiser	550.00	2	3
*	NULL	NULL	NULL	NULL	NULL

	ProductID	ProductName	Price	ManufacturerID	CategoryID
▶	4	Cycling Gloves	29.99	4	4
	5	Mountain Helmet Pro	49.99	2	4
	8	Bike Repair Kit	59.99	5	4
	7	City Cruiser 300	450.00	3	3
*	NULL	NULL	NULL	NULL	NULL

Customer table

As CustomerName is used within the following queries, a secondary index has been implemented to it to improve the performance and retrieval speeds.

When searching for Customer details, data retrieval could be sped up using queries to filter by ID or name, which are both demonstrated and shown below.

These queries will be useful s management may need to access a specific customer’s details to contact them about any changes or updates to their order status.

```
-- Customer table -----
-- Secondary indexes
CREATE INDEX idx_CustomerName ON Customer(CustomerName);

-- A query to view a Customer when given a CustomerID
SELECT *
FROM Customer
WHERE CustomerID = 5
LIMIT 1;

-- A query to view a Customer when given a CustomerName
SELECT *
FROM Customer
WHERE CustomerName = 'John Doe'
LIMIT 1;
```

	CustomerID	CustomerName	CustomerEmail	CustomerPhoneNum
▶	5	Paul Brown	paul.brown@example.com	07722334455
✱	NULL	NULL	NULL	NULL

	CustomerID	CustomerName	CustomerEmail	CustomerPhoneNum
▶	1	John Doe	john.doe@example.com	07812345678
✱	NULL	NULL	NULL	NULL

OrderInfo and OrderLine table

Views were created using the OrderInfo and OrderLine tables so that the total price of each OrderLine record and the total price of each OrderInfo record could be calculated. It was done using this method as so to maintain 3rd form normalisation within these tables, as the price was already stored within the Product table, so repeating the data within the OrderInfo and OrderLine table would mean that 1NF criteria was no longer met.

As OrderInfoID and ProductID from the OrderLine table and CustomerID from the OrderInfo table are used within the following queries, secondary indexes have been implemented to them to improve the performance and retrieval speeds.

The OrderLineWithPrice view is created to show all relevant OrderLine information as well as the Product's price and the price multiplied by quantity to show the TotalLinePrice. This is done using a join between the OrderLine and Product tables.

The OrderInfoWithPrice view is similar, however calculated the total price of each OrderInfo record. This is done using a join between the OrderInfo and Product tables, as well as grouping the OrderInfoID's together [8].

```
-- Views -----
-- Secondary indexes
CREATE INDEX idx_OrderInfoID ON OrderLine(OrderInfoID);
CREATE INDEX idx_OrderLineProductID ON OrderLine(ProductID);
CREATE INDEX idx_OrderInfoCustomerID ON OrderInfo(CustomerID);

-- Creation of the View on the OrderLine table -----
-- SELECTS all relevant information from the OrderLine table, including the multiplication of quantity and price to give the TotalLinePrice
CREATE VIEW OrderLineWithPrice AS
SELECT OrderLine.OrderLineID, OrderLine.OrderInfoID, OrderLine.ProductID, OrderLine.Quantity, Product.Price AS ProductPrice, (OrderLine.Quantity * Product.Price) AS TotalLinePrice
FROM OrderLine
JOIN Product ON OrderLine.ProductID = Product.ProductID;

-- Creation of the View on the OrderInfo table -----
-- SELECTS all relevant information from the OrderLine table, including the sum of all values in the TotalLinePrice column to give TotalPrice
CREATE VIEW OrderInfoWithTotalPrice AS
SELECT OrderInfo.OrderInfoID, OrderInfo.CustomerID, OrderInfo.OrderDate, SUM(OrderLineWithPrice.TotalLinePrice) AS TotalPrice
FROM OrderInfo
JOIN OrderLineWithPrice ON OrderInfo.OrderInfoID = OrderLineWithPrice.OrderInfoID
GROUP BY OrderInfo.OrderInfoID, OrderInfo.CustomerID, OrderInfo.OrderDate;
```

Queries were then performed on these views to swiftly retrieve frequently searched information. These following queries will be useful as price information is always relevant and very important. The ability to quickly and efficiently access price information will greatly benefit the managers within Adventure Works Cycles.

```
-- Queries on the created views -----
-- A query to view all orders, with prices
SELECT *
FROM OrderInfoWithTotalPrice;

-- A query to view all orders, with prices, placed by a specific customerID
SELECT *
FROM OrderInfoWithTotalPrice
WHERE CustomerID = 1;

-- A query to view all orderLines, with prices
SELECT *
FROM OrderLineWithPrice
ORDER BY OrderInfoID ASC, ProductID ASC;
```

This query on the OrderInfoWithTotalPrice view shows each OrderInfo record with the total price of that order.

	OrderInfoID	CustomerID	OrderDate	TotalPrice
▶	1	3	2024-12-01 10:30:00	2649.98
	2	5	2024-12-02 14:45:00	2459.95
	3	1	2024-12-03 13:15:00	1649.99
	4	4	2024-12-04 09:30:00	2449.97
	5	2	2024-12-05 16:25:00	1349.99
	6	5	2024-12-06 11:55:00	1119.97
	7	3	2024-12-07 17:30:00	3079.97
	8	1	2024-12-08 13:05:00	1599.98
	9	4	2024-12-09 15:45:00	1779.96

This query on the OrderInfoWithTotalPrice view shows each OrderInfo record placed by a specified CustomerID with the total price of that order.

	OrderInfoID	CustomerID	OrderDate	TotalPrice
▶	3	1	2024-12-03 13:15:00	1649.99
	8	1	2024-12-08 13:05:00	1599.98

This query on the OrderLineWithTotalPrice view shows each OrderLine record with the total price of that order.

	OrderLineID	OrderInfoID	ProductID	Quantity	ProductPrice	TotalLinePrice
▶	1	1	1	2	999.99	1999.98
	2	1	3	1	650.00	650.00
	3	2	2	3	799.99	2399.97
	4	2	4	2	29.99	59.98
	6	3	1	1	999.99	999.99
	5	3	3	1	650.00	650.00
	7	4	5	1	49.99	49.99
	8	4	6	2	1199.99	2399.98
	9	5	7	1	450.00	450.00
	10	5	9	1	899.99	899.99
	12	6	1	1	999.99	999.99
	11	6	8	2	59.99	119.98
	13	7	3	1	650.00	650.00
	14	7	4	1	29.99	29.99
	15	7	6	2	1199.99	2399.98
	16	8	2	2	799.99	1599.98
	20	9	2	1	799.99	799.99
	21	9	4	1	29.99	29.99
	17	9	5	1	49.99	49.99
	18	9	9	1	899.99	899.99

Feedback table

As ProductID and Rating are used within the following queries, secondary indexes have been implemented to them to improve the performance and retrieval speeds.

The queries designed and created for use on the Feedback table can be seen in the following screenshots.

The first query is used to view all Feedback records for a specific given ProductID, which could be used by management to decide whether or not to continue selling that product.

The second query is used to display the average rating value for all products, as well as how many reviews each product has received, which could be used to gauge the overall performance and feedback for each product and compare between them.

```
-- Queries on the Feedback table -----
-- Secondary Indexes
CREATE INDEX idx_FeedbackProductID ON Feedback(ProductID);
CREATE INDEX idx_Rating ON Feedback(Rating);

-- A query to view all feedback given for a specific ProductID
SELECT Feedback.FeedbackID, Feedback.ProductID, Product.ProductName, Product.Price, Customer.CustomerID, Customer.CustomerName, Feedback.Rating, Feedback.Comments
FROM Feedback
JOIN Product ON Feedback.ProductID = Product.ProductID
JOIN Customer ON Feedback.CustomerID = Customer.CustomerID
WHERE Feedback.ProductID = 3;

-- A query to view the average rating of each product
SELECT ProductID, Round(AVG(Rating), 2) AS AvgRating, COUNT(Rating) AS TotalReviews
FROM Feedback
GROUP BY ProductID
ORDER BY AvgRating DESC;
```

	FeedbackID	ProductID	ProductName	Price	CustomerID	CustomerName	Rating	Comments
▶	1	3	Hybrid Trail 700	650.00	3	Mike Johnson	4	Hybrid Trail 700 is a great bike for both roads and light trails.
	4	3	Hybrid Trail 700	650.00	1	John Doe	4	Hybrid Trail 700 is a solid choice for all-around cycling. Good value for money.

	ProductID	AvgRating	TotalReviews
▶	1	5.00	1
	2	5.00	2
	5	5.00	1
	6	5.00	1
	9	5.00	1
	3	4.00	2
	8	4.00	1
	4	3.50	2
	7	3.00	1

References

- [M. Fayard, "What is Third Normal Form (3NF)?," Datacamp, 18 11 2024. [Online].
1 Available: <https://www.datacamp.com/tutorial/third-normal-form>. [Accessed 05 01 2025].
]
- [A. Zanini, "Database Transactions 101: The Essential Guide," DbVisualizer, 14 02 2023.
2 [Online]. Available: <https://www.dbvis.com/thetable/database-transactions-101-the-essential-guide/#:~:text=A%20database%20transaction%20is%20a,operations%20performed%20within%20a%20DBMS..> [Accessed 05 01 2025].
]
- ["Difference Between COMMIT and ROLLBACK in SQL," GeeksForGeeks.org, 19 12
3 2024. [Online]. Available: <https://www.geeksforgeeks.org/difference-between-commit-and-rollback-in-sql/>. [Accessed 05 01 2025].
]
- ["SQL TRANSACTIONS," GeeksForGeeks.org, 03 09 2024. [Online]. Available:
4 <https://www.geeksforgeeks.org/sql-transactions/>. [Accessed 05 01 2025].
]
- ["sql try/catch rollback/commit - preventing erroneous commit after rollback," Stack
5 Overflow, 01 07 2018. [Online]. Available:
] <https://stackoverflow.com/questions/25146656/sql-try-catch-rollback-commit-preventing-erroneous-commit-after-rollback>. [Accessed 05 01 2025].
]
- [L. Ballejos, "What is a Database Query?," NinjaOne, 01 02 2024. [Online]. Available:
6 <https://www.ninjaone.com/it-hub/it-service-management/what-is-a-database-query/>.
] [Accessed 05 01 2025].
- ["Secondary indexes," Oracle Help CEnter, [Online]. Available:
7 [https://docs.oracle.com/cd/E17275_01/html/programmer_reference/am_second.html#:~:text=A%20secondary%20index%2C%20put%20simply,the%20usual%20\(primary\)%20key..](https://docs.oracle.com/cd/E17275_01/html/programmer_reference/am_second.html#:~:text=A%20secondary%20index%2C%20put%20simply,the%20usual%20(primary)%20key..)
] [Accessed 05 01 2025].
- ["SQL SUM() Function," W3Schools, [Online]. Available:
8 https://www.w3schools.com/sql/sql_sum.asp. [Accessed 05 01 2025].
]

Appendix

Prompts used within Generative AI / ChatGPT:

Prompt 1 -----

“Are these tables normalised to 3rd normal form?:

-- Creates and assigns the database as active

CREATE DATABASE assignment;

USE assignment;

-- Creates the Manufacturer table to store Manufacturer's names and contact details

```
CREATE TABLE Manufacturer (  
    ManufacturerID INT AUTO_INCREMENT PRIMARY KEY,  
    ManufacturerName CHAR(100) NOT NULL,  
    ManufacturerEmail CHAR(100),  
    ManufacturerPhoneNum CHAR(15)  
);
```

-- Creates the Category table to store product type classification information

```
CREATE TABLE Category (  
    CategoryID INT AUTO_INCREMENT PRIMARY KEY,  
    CategoryName CHAR(100) NOT NULL  
);
```

-- Creates the Product table to store the information on the items available for sale

-- There can be many products from one category

-- There can be many products sold by one manufacturer

```
CREATE TABLE Product (  
    ProductID INT AUTO_INCREMENT PRIMARY KEY,  
    ProductName CHAR(100) NOT NULL,  
    ProductDescription CHAR(200),  
    Price DECIMAL(10,2),  
    CategoryID INT,
```

ManufacturerID INT

);

-- Creates the Customer table to store the information of the individuals or organizations that purchase products

CREATE TABLE Customer (

CustomerID INT AUTO_INCREMENT PRIMARY KEY,

CustomerName CHAR(100) NOT NULL,

CustomerEmail CHAR(100),

CustomerPhoneNum CHAR(15)

);

-- Creates the OrderInfo table to store the overall information of each order

-- There can be many orders placed by one customer

CREATE TABLE OrderInfo (

OrderInfoID INT AUTO_INCREMENT PRIMARY KEY,

CustomerID INT,

OrderDate DATETIME

);

-- Creates the OrderLine table to store each individual product purchase and what order it belongs to

-- There can be many OrderLines assigned to one OrderInfo

CREATE TABLE OrderLine (

OrderLineID INT AUTO_INCREMENT PRIMARY KEY,

OrderInfoID INT,

ProductID INT,

Quantity INT NOT NULL

);

-- Creates the Feedback table to store feedback and ratings provided by the customers

-- There can be many sets of feedback given by one customer

-- There can be many sets of feedback given for one product

```
CREATE TABLE Feedback (  
    FeedbackID INT AUTO_INCREMENT PRIMARY KEY,  
    CustomerID INT,  
    ProductID INT,  
    Rating INT NOT NULL,  
    Comments CHAR(250)  
);
```

Prompt 2 -----

“Create sample data for the following tables, ensuring that records are aligned and correctly matched between the OrderInfo, OrderLine, Feedback tables (Make sure that any OrderLine records match up with the OrderInfo records and that any feedback is only given by customers and for products that they’ve bought etc.):

-- Creates and assigns the database as active

```
CREATE DATABASE assignment;  
USE assignment;
```

-- Creates the Manufacturer table to store Manufacturer’s names and contact details

```
CREATE TABLE Manufacturer (  
    ManufacturerID INT AUTO_INCREMENT PRIMARY KEY,  
    ManufacturerName CHAR(100) NOT NULL,  
    ManufacturerEmail CHAR(100),  
    ManufacturerPhoneNum CHAR(15)  
);
```

-- Creates the Category table to store product type classification information

```
CREATE TABLE Category (  
    CategoryID INT AUTO_INCREMENT PRIMARY KEY,  
    CategoryName CHAR(100) NOT NULL  
);
```

- Creates the Product table to store the information on the items available for sale
- There can be many products from one category
- There can be many products sold by one manufacturer

```
CREATE TABLE Product (  
    ProductID INT AUTO_INCREMENT PRIMARY KEY,  
    ProductName CHAR(100) NOT NULL,  
    ProductDescription CHAR(200),  
    Price DECIMAL(10,2),  
    CategoryID INT,  
    ManufacturerID INT  
);
```

- Creates the Customer table to store the information of the individuals or organizations that purchase products

```
CREATE TABLE Customer (  
    CustomerID INT AUTO_INCREMENT PRIMARY KEY,  
    CustomerName CHAR(100) NOT NULL,  
    CustomerEmail CHAR(100),  
    CustomerPhoneNum CHAR(15)  
);
```

- Creates the OrderInfo table to store the overall information of each order
- There can be many orders placed by one customer

```
CREATE TABLE OrderInfo (  
    OrderInfoID INT AUTO_INCREMENT PRIMARY KEY,  
    CustomerID INT,  
    OrderDate DATETIME  
);
```

- Creates the OrderLine table to store each individual product purchase and what order it belongs to
- There can be many OrderLines assigned to one OrderInfo

```
CREATE TABLE OrderLine (
```

```
        OrderLineID INT AUTO_INCREMENT PRIMARY KEY,  
        OrderInfoID INT,  
        ProductID INT,  
        Quantity INT NOT NULL  
    );
```

-- Creates the Feedback table to store feedback and ratings provided by the customers

-- There can be many sets of feedback given by one customer

-- There can be many sets of feedback given for one product

```
CREATE TABLE Feedback (  
    FeedbackID INT AUTO_INCREMENT PRIMARY KEY,  
    CustomerID INT,  
    ProductID INT,  
    Rating INT NOT NULL,  
    Comments CHAR(250)  
);
```

-- Assigns CategoryID and ManufacturerID as Foreign keys in the Product table

```
ALTER TABLE Product  
ADD FOREIGN KEY (CategoryID) REFERENCES Category(CategoryID),  
ADD FOREIGN KEY (ManufacturerID) REFERENCES Manufacturer(ManufacturerID);
```

-- Assigns CustomerID as Foreign keys in the OrderInfo table

```
ALTER TABLE OrderInfo  
ADD FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID);
```

-- Assigns OrderInfoID and ProductID as Foreign keys in the OrderLine table

```
ALTER TABLE OrderLine  
ADD FOREIGN KEY (OrderInfoID) REFERENCES OrderInfo(OrderInfoID),  
ADD FOREIGN KEY (ProductID) REFERENCES Product(ProductID);
```

-- Assigns CustomerID and ProductID as Foreign keys in the Feedback table

ALTER TABLE Feedback

ADD FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID),

ADD FOREIGN KEY (ProductID) REFERENCES Product(ProductID);

-- Assigns a check constraint to the Product table, ensuring that Price is a positive value as a price cannot be negative

ALTER TABLE Product

ADD CONSTRAINT checkPrice CHECK (Price > 0);

-- Assigns a check constraint to the OrderLine table, ensuring that Quantity a positive value as a quantity can not be negative

ALTER TABLE OrderLine

ADD CONSTRAINT checkQuantity CHECK (Quantity > 0);

-- Assigns a check constraint to the Feedback table, ensuring that Rating is a value in the range 1-5 inclusive, to act as a 5 star rating

ALTER TABLE Feedback

ADD CONSTRAINT checkRating CHECK (Rating BETWEEN 1 AND 5);

Prompt 3 -----

"How should I group this view:

CREATE VIEW OrderInfoWithTotalPrice AS

SELECT OrderInfo.OrderInfoID, OrderInfo.CustomerID, OrderInfo.OrderDate,
SUM(OrderLineWithPrice.TotalLinePrice) AS TotalPrice

FROM OrderInfo

JOIN OrderLineWithPrice ON OrderInfo.OrderInfoID =
OrderLineWithPrice.OrderInfoID;"