

2311568 - Database Assignment Report

by Nate Fox

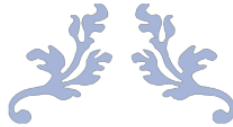
Submission date: 07-Jan-2025 11:17AM (UTC+0000)

Submission ID: 247465904

File name: 49385_Nate_Fox_2311568_-_Database_Assignment_Report_621315_1979327765.docx (4.79M)

Word count: 3005

Character count: 16949



DATABASES AND APPLICATION DEVELOPMENT

Assignment Title: AdventureWorks Cycles' Database

Design & Implementation

Student ID: 2311568

Nate Fox



Contents

Introduction	3
Database Creation	4
Entity Relationship Diagram	4
ERD Design Justification	5
Database Implementation	6
SQL Code Screenshots	6
SQL Query Tables	13
SQL Queries for Testing	16
Query 1	16
Query 2	17
Query 3	18
Query 4	19
Query 5	20
Transaction Management	21
Commit	21
Rollback	23
Database Performance Indexes	25
Report Conclusion	26
References	27



Introduction

AdventureWorks Cycles, known for its range of bicycles and accessories, aims to expand its business by launching an online platform for browsing, customizing, and purchasing products. A robust database is essential to manage customer and manufacturer information, track orders and payments, and store product details efficiently.

This report outlines the design and development of the database, beginning with an Entity Relationship Diagram (ERD) illustrating table connections. It also includes the SQL code used for table creation, sample data insertion, and testing. Additionally, the report covers database optimization and testing to ensure it meets the company's requirements.



Database Creation

Entity Relationship Diagram

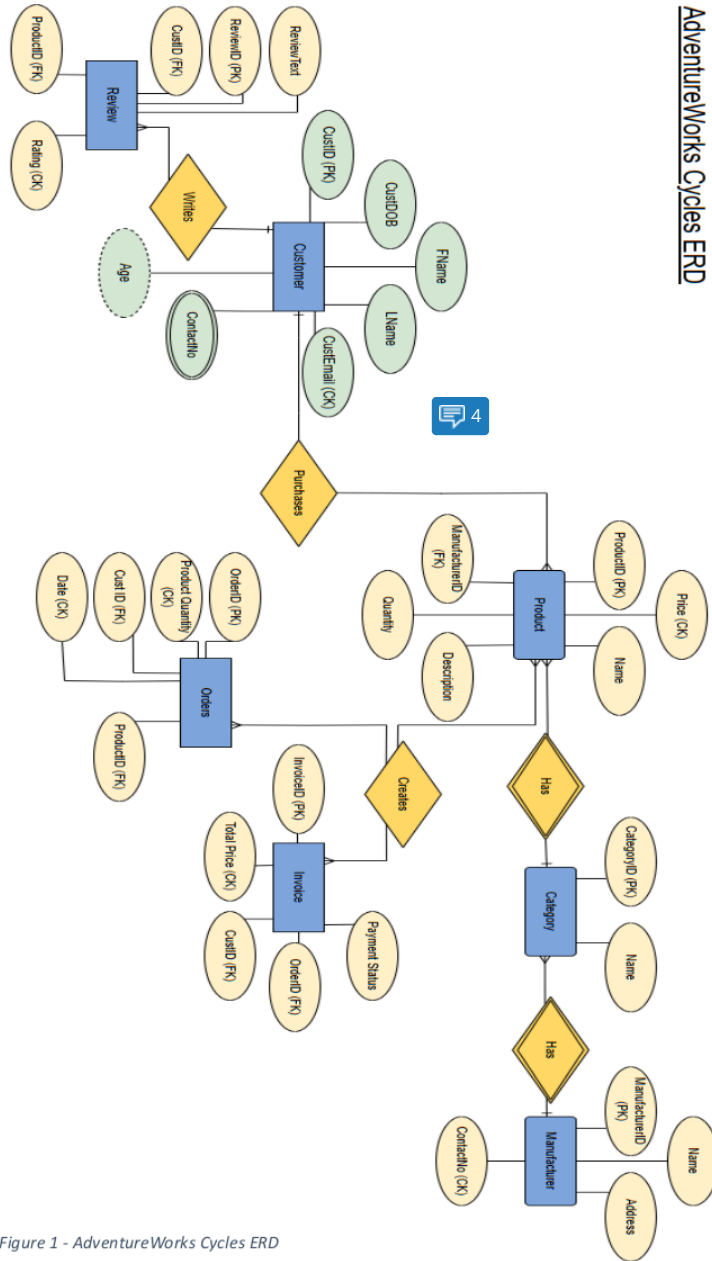


Figure 1 - AdventureWorks Cycles ERD



ERD Design Justification

The Entity Relationship Diagram (ERD) for AdventureWorks Cycles show how the different parts of the database are connected to support key processes like managing customers, roles, products and reviews [1][2][3].

Customer Table:

The customer table stores key details about each customer, such as their name, date of birth, and contact information. The primary key (CustID) ensures each customer is unique. The table's relationships indicate that a customer can place multiple orders and write product reviews.

Product Table:

This table contains information about each product, including its name, description, price, and stock quantity. It links to the manufacturer and category tables via foreign keys (ManufacturerID and CategoryID), indicating that each product belongs to a specific category and is produced by a manufacturer.

Manufacturer and Category Tables:

The manufacturer table tracks details about companies that produce products, such as their name, address, and contact number. The category table organizes products into groups, such as "Road Bikes" or "Accessories," to facilitate product filtering.

Orders Table:

The orders table records customer purchases, specifying the product, quantity, and date of the order. It connects to the customer and product tables to ensure each order is linked to valid entries. Check constraints, such as ensuring quantities are above zero, help maintain data integrity.

Invoice Table:

The invoice table logs payment details for each order, including the total price and payment status. It links to the order and customer tables, making it clear which customer the payment relates to.

Review Table:

The review table enables customers to leave feedback on purchased products, with fields for a rating (between 1 and 5) and review text. Relationships ensure that reviews can only be created for valid customers and products.

Normalization:

The database is normalized to 3rd normal form (3NF), meaning each piece of information is stored in the appropriate table without redundancy. For example, customer details are not repeated in the orders table. This minimizes data duplication and ensures each table focuses on a single topic, making the database more efficient and easier to maintain [4].

Referential Integrity:

Foreign keys ensure valid relationships, such as linking orders to valid products and customers. Rules like ON DELETE CASCADE ensure that related records (e.g. orders) are handled correctly when a primary record (e.g. a customer's account) is deleted.

Each table in the Entity Relationship Diagram (ERD) serves a distinct purpose and connects logically to others, enabling the database to handle tasks like tracking orders, storing product information, and managing customer feedback while remaining well-organized, accurate, and easy to query.



Database Implementation

SQL Code Screenshots

This section will provide screenshot snippets, along with a description to each.

```
1 • Create Database AdventureWorks; -- Creates the Database.
```

Figure 2 – The Create Database AdventureWorks; command initializes the database where all the tables and data will be stored. This is the first step in setting up the database.

```
3 • Use AdventureWorks; -- This tells the code to use the database, next need to
4      -- add tables.
5
6      -- The creation of the tables have been based on logical dependencies than
7      -- strictly following the flow of the ERD.
```

Figure 3 – The Use AdventureWorks; command sets the context to the AdventureWorks Database, ensuring that the following SQL commands apply to this specific database.

```
8      -- Since other tables such as orders and reviews depend on CustID the customer
9      -- table is made first.
10 • CREATE TABLE Customer (
11     CustID INT PRIMARY KEY,
12     FName VARCHAR(50) NOT NULL,
13     LName VARCHAR(50) NOT NULL,
14     CustDOB DATE NOT NULL,
15     CustEmail VARCHAR(100) UNIQUE NOT NULL, -- Unique: If a value already exists
16                                           -- in the row the database will throw
17                                           -- out an error.
18     ContactNo VARCHAR(15) CHECK (ContactNo LIKE '+%') -- + ensures the the value
19                                           -- starts with a + character.
20 );
```

Figure 4 – The customer table stores customer details such as name, date of birth, email, and contact number. The primary key (CustID) ensures each customer has a unique identifier, while a unique constraint on CustEmail prevents duplicate entries. A check constraint ensures that ContactNo starts with a "+" symbol to follow the required format. CustID also serves as a foreign key in related tables to maintain referential integrity.



```

22  -- The manufacturer table and category tables are next to ensure their respective
23  -- ID's are available for the product table.
24  CREATE TABLE Manufacturer (
25      ManufacturerID INT PRIMARY KEY,
26      Name VARCHAR(100) NOT NULL,
27      Address VARCHAR(250) NOT NULL,
28      ContactNo VARCHAR(15) CHECK (ContactNo LIKE '+%[0-9]%')
29  );
30
31  CREATE TABLE Category (
32      CategoryID INT PRIMARY KEY,
33      Name VARCHAR(100) NOT NULL
34  );

```

Figure 5 – The manufacturer table stores manufacturer details, including name, address, and contact number, with a primary key (ManufacturerID) to uniquely identify each manufacturer. A check constraint ensures the contact number starts with a "+". The category table classifies products into categories like "Mountain Bikes" or "Accessories," with CategoryID as the primary key to ensure category uniqueness.

```

36  -- As both ManufacturerID and CategoryID are foreign keys within the Products table
37  -- it's logical to add the products table next.
38  CREATE TABLE Product (
39      ProductID INT PRIMARY KEY,
40      Name VARCHAR(100) NOT NULL,
41      Description TEXT,
42      Price DECIMAL (10,2) CHECK (Price > 0), -- The DECIMAL ensures the correct layout
43                                              -- and the price constraint ensures no
44                                              -- negative value's or '0' are entered.
45      Quantity INT CHECK (Quantity >= 0),
46      ManufacturerID INT,
47      CategoryID INT,
48      FOREIGN KEY (ManufacturerID) REFERENCES Manufacturer(ManufacturerID) ON DELETE CASCADE,
49                                              -- Ensuring referential integrity
50      FOREIGN KEY (CategoryID) REFERENCES Category(CategoryID) ON DELETE SET NULL
51      -- ON DELETE SET NULL will set the foreign key to NULL
52      -- without deleting the product unlike ON DELETE CASCADE will
53      -- delete all child records when a category is deleted.
54  );

```

Figure 6 – The product table stores product details such as name, description, price, and stock quantity. The primary key (ProductID) ensures each product is unique. Check constraints enforce a price greater than 0 and a non-negative stock quantity. Foreign keys link ManufacturerID and CategoryID to the manufacturer and category tables, ensuring products are associated with valid entries. The ON DELETE CASCADE and ON DELETE SET NULL options handle deletions to maintain data consistency.



```

56 -- Orders table is next as it requires Customer and product for foreign key relationships.
57 • CREATE TABLE Orders (
58     OrderID INT PRIMARY KEY,
59     CustID INT,
60     ProductID INT,
61     ProductQuantity INT CHECK (ProductQuantity > 0), -- Representing stock count or number of
62                                                         -- items available. Check constraints prevents
63                                                         -- invalid numbers being added.
64     Date DATE NOT NULL,
65     FOREIGN KEY (CustID) REFERENCES Customer (CustID) ON DELETE CASCADE,
66     FOREIGN KEY (ProductID) REFERENCES Product (ProductID) ON DELETE CASCADE -- Ensuring referential
67                                                         -- integrity
68 );

```

Figure 7 – The orders table tracks customer purchases. It includes fields for OrderID (the primary key), CustID (foreign key to customer), and ProductID (foreign key to product). A check constraint on ProductQuantity ensures that only positive values are accepted. The ON DELETE CASCADE rule ensures that when a customer or product is deleted, related orders are automatically removed to maintain integrity.

```

70 -- The invoice table relates to the order table through OrderID and depends on customer for CustID so
71 -- comes next.
72 • CREATE TABLE Invoice (
73     InvoiceID INT PRIMARY KEY,
74     OrderID INT,
75     CustID INT,
76     TotalPrice DECIMAL (10,2) CHECK (TotalPrice > 0),
77     PaymentStatus VARCHAR (50),
78     FOREIGN KEY (OrderID) REFERENCES Orders(OrderID) ON DELETE CASCADE,
79     FOREIGN KEY (CustID) REFERENCES Customer(CustID) ON DELETE CASCADE
80 );

```

Figure 8 – The Invoice table stores payment-related details, such as the total price and payment status. The InvoiceID is the primary key, ensuring each invoice is unique. Foreign keys (OrderID and CustID) maintain references to the orders and customer tables. The ON DELETE CASCADE option ensures invoices are removed if the associated order or customer is deleted. The check constraint on TotalPrice ensures the total is a positive value.

```

82 -- The review table comes next as it depends on both customer and product.
83 • CREATE TABLE Review (
84     ReviewID INT PRIMARY KEY,
85     CustID INT,
86     ProductID INT,
87     Rating INT CHECK (Rating BETWEEN 1 AND 5), -- This provides a rating for service/product between 1-5.
88     ReviewText TEXT,
89     FOREIGN KEY (CustID) REFERENCES Customer(CustID) ON DELETE CASCADE, -- ON DELETE CASCADE ensures all
90                                                         -- tables are updated if a change
91                                                         -- is made.
92     FOREIGN KEY (ProductID) REFERENCES Product(ProductID) ON DELETE CASCADE
93 );

```

Figure 9 – The review table records customer feedback for products, including a rating and review text. The primary key is ReviewID. A check constraint ensures that the rating is between 1 and 5. Foreign keys (CustID and ProductID) ensure reviews correspond to valid customers and products. The ON DELETE CASCADE option ensures reviews are removed if a related customer or product is deleted.



```

95 -- All tables have now been added to the database. x10 sample data will now be added to each table. Starting
96 -- with customer table.
97 • INSERT INTO Customer (CustID, FName, LName, CustDOB, CustEmail, ContactNo)
98 VALUES
99 (1, 'John', 'Doe', '1990-05-15', 'john.doe@example.com', '+757688882'),
100 (2, 'Jane', 'Smith', '1985-10-20', 'jane.smith@example.com', '+757688883'),
101 (3, 'Michael', 'Johnson', '2000-01-10', 'michael.j@example.com', '+757688884'),
102 (4, 'Emily', 'Davis', '1995-07-25', 'emily.davis@example.com', '+757688885'),
103 (5, 'Daniel', 'Brown', '1987-03-14', 'daniel.brown@example.com', '+757688886'),
104 (6, 'Sophia', 'Wilson', '1993-06-22', 'sophia.wilson@example.com', '+757688887'),
105 (7, 'Lucas', 'Taylor', '1999-12-30', 'lucas.taylor@example.com', '+757688888'),
106 (8, 'Olivia', 'Anderson', '2001-09-11', 'olivia.anderson@example.com', '+757688889'),
107 (9, 'Ethan', 'Moore', '1992-02-05', 'ethan.moore@example.com', '+757688890'),
108 (10, 'Ava', 'Martin', '1989-11-18', 'ava.martin@example.com', '+757688891');

```

Figure 10 -The insertion of 10 sample customers into the customer table for testing purposes. Sample data includes names, emails and formatted contact numbers.

```

110 • ALTER TABLE Customer DROP CONSTRAINT customer_chk_1; -- I had an issue trying to add the customer information
111 -- into the customer table.
112 -- I had to drop the initial constraint and add a new one
113 -- to remove the violation.
114 • ALTER TABLE Customer
115 ADD CONSTRAINT customer_chk_1 CHECK (ContactNo LIKE '+%'); -- Customer sample data been added.

```

Figure 11 -An issue with the initial check constraint for ContactNo in and customer table is resolved by dropping the existing constraint and adding a corrected one. This ensures that the data follows the correct format.

```

117 -- Adding sample data into the manufacturer table.
118 • INSERT INTO Manufacturer (ManufacturerID, Name, Address, ContactNo)
119 VALUES
120 (1, 'CyclePros', '123 Main Street, London', '+447700900123'),
121 (2, 'TrailBlazers', '456 Elm Avenue, Manchester', '+447812345678'),
122 (3, 'SpeedRiders', '789 Oak Lane, Birmingham', '+447934567890'),
123 (4, 'BikeMasters', '321 Pine Road, Leeds', '+447945678901'),
124 (5, 'UrbanCycles', '654 Cedar Avenue, Bristol', '+447856789012'),
125 (6, 'AdventureGear', '987 Spruce Boulevard, Glasgow', '+447967890123'),
126 (7, 'VelocityWorks', '159 Birch Drive, Edinburgh', '+447978901234'),
127 (8, 'HighlandCycles', '753 Maple Court, Inverness', '+447989012345'),
128 (9, 'GearUp', '951 Walnut Plaza, Cardiff', '+447990123456'),
129 (10, 'RiderTech', '852 Cherry Loop, Liverpool', '+447901234567');

```

Figure 12 – Ten sample manufacturers are inserted into the manufacturer table, including names, addresses and contact numbers for each company.



```

131 -- I have run into a similar issue with a 'manufacturer_chk_1' violation. I'll copy the step I did for the
132 -- customer table.
133
134 • ALTER TABLE Manufacturer DROP CONSTRAINT manufacturer_chk_1;
135 • ALTER TABLE Manufacturer
136   ADD CONSTRAINT manufacturer_chk_1 CHECK (ContactNo LIKE '+%'); -- This solution worked and manufacturer
137                               -- sample data has been added.

```

Figure 13 – A similar contact number constraint issue in the manufacturer table is resolved by dropping and readding the check constraint.

```

139 -- Next adding Category sample data into the database.
140 • INSERT INTO Category (CategoryID, Name)
141   VALUES
142   (1, 'Mountain Bikes'),
143   (2, 'Road Bikes'),
144   (3, 'Accessories'),
145   (4, 'Electric Bikes'),
146   (5, 'Hybrid Bikes'),
147   (6, 'Kids Bikes'),
148   (7, 'Cruisers'),
149   (8, 'Fat Bikes'),
150   (9, 'Folding Bikes'),
151   (10, 'Gravel Bikes');

```

Figure 14 – Ten sample categories are added to the category table, covering different types of products such as “Road Bikes” and “Accessories”.

```

153 -- Next adding product sample data into the database.
154 • INSERT INTO Product (ProductID, Name, Description, Price, Quantity, ManufacturerID, CategoryID)
155   VALUES
156   (1, 'Trail Blazer 3000', 'High-performance mountain bike', 1200.00, 10, 1, 1),
157   (2, 'Road Warrior Pro', 'Lightweight road bike', 1500.00, 5, 2, 2),
158   (3, 'Cycling Helmet', 'Safety helmet for all riders', 100.00, 50, 2, 3),
159   (4, 'Bike Lights', 'Front and rear LED bike lights', 40.00, 100, 1, 3),
160   (5, 'Electric Commuter', 'Electric bike for city commuting', 2200.00, 7, 3, 4),
161   (6, 'Hybrid Explorer', 'Versatile bike for road and trails', 900.00, 12, 4, 5),
162   (7, 'Kid Cruiser', 'Safe and fun bike for kids', 250.00, 20, 5, 6),
163   (8, 'Beach Rider', 'Comfortable cruiser for beach rides', 500.00, 8, 6, 7),
164   (9, 'Fat Tire Beast', 'Heavy-duty fat tire bike', 1700.00, 6, 7, 8),
165   (10, 'Compact Fold', 'Easy-to-store folding bike', 800.00, 15, 8, 9);

```

Figure 15 – Ten sample products are inserted into the product table. Each product is linked to a manufacturer and a category using their respective foreign keys.



```

167 -- Next adding order sample data into the database.
168 • INSERT INTO Orders (OrderID, CustID, ProductID, ProductQuantity, Date)
169 VALUES
170 (1, 1, 1, 2, '2024-12-01'),
171 (2, 2, 2, 1, '2024-12-02'),
172 (3, 3, 3, 3, '2024-12-03'),
173 (4, 4, 4, 1, '2024-12-04'),
174 (5, 5, 5, 1, '2024-12-05'),
175 (6, 6, 6, 2, '2024-12-06'),
176 (7, 7, 7, 1, '2024-12-07'),
177 (8, 8, 8, 1, '2024-12-08'),
178 (9, 9, 9, 1, '2024-12-09'),
179 (10, 10, 10, 2, '2024-12-10');

```

Figure 16 – Ten sample orders are added to the orders table.

```

181 -- Next adding invoice sample data to the database.
182 • INSERT INTO Invoice (InvoiceID, OrderID, CustID, TotalPrice, PaymentStatus)
183 VALUES
184 (1, 1, 1, 2400.00, 'Paid'),
185 (2, 2, 2, 1500.00, 'Paid'),
186 (3, 3, 3, 300.00, 'Pending'),
187 (4, 4, 4, 40.00, 'Paid'),
188 (5, 5, 5, 2200.00, 'Paid'),
189 (6, 6, 6, 1800.00, 'Paid'),
190 (7, 7, 7, 250.00, 'Pending'),
191 (8, 8, 8, 500.00, 'Paid'),
192 (9, 9, 9, 1700.00, 'Pending'),
193 (10, 10, 10, 1600.00, 'Paid');

```

Figure 17 - Ten sample invoices are inserted into the orders table, showing payment details for each order, including total prices, and payment statutes ("Paid" or "Pending").

```

197 • SELECT * FROM Customer; -- Checking to ensure data is showing in customer table.
198 • SELECT * FROM Manufacturer; -- Checking to ensure data is showing in manufacturer table.
199 • SELECT * FROM Category; -- Checking to ensure data is showing in category table.
200 • SELECT * FROM Product; -- Checking to ensure data is showing in product table.
201 • SELECT * FROM Orders; -- Checking to ensure data is showing in orders table.
202 • SELECT * FROM Invoice; -- Checking to ensure data is showing in Invoice table.
203 • SELECT * FROM Review; -- Checked to ensure table was working will add some sample data to
204 -- the table for paid orders.

```

Figure 18 – SELECT * FROM queries were tested to display the contents of the tables, confirming that the sample data has been correctly inserted into the customer, manufacturer, product, orders, review and invoice tables.



```
206 -- Adding sample data to review table for items purchased
207 • INSERT INTO Review (ReviewID, CustID, ProductID, Rating, ReviewText)
208 VALUES
209 (1, 1, 1, 5, 'Fantastic bike, exceeded expectations!'),
210 (2, 2, 2, 4, 'Great bike, but a bit pricey.'),
211 (3, 4, 4, 5, 'Bike lights are super bright and long-lasting.'),
212 (4, 5, 5, 5, 'The electric commuter bike is a game changer!'),
213 (5, 6, 6, 4, 'Hybrid bike is versatile and sturdy.'),
214 (6, 8, 8, 3, 'Comfortable, but not ideal for long rides.'),
215 (7, 10, 10, 5, 'Folding bike is very convenient and compact.');
```

Figure 19 – Seven sample reviews are added to the review table, each containing a customer's rating and review text for purchased products.

```
217 • Select * FROM Review; -- Checked to ensure data for reviews have been pulled through for
```

Figure 20 – This SELECT * FROM Review; query retrieves all the data from the review table to verify that the sample reviews have been successfully inserted.



SQL Query Tables

Customer Table

	CustID	FName	LName	CustDOB	CustEmail	ContactNo
▶	1	John	Doe	1990-05-15	john.doe@example.com	+757688882
	2	Jane	Smith	1985-10-20	jane.smith@example.com	+757688883
	3	Michael	Johnson	2000-01-10	michael.j@example.com	+757688884
	4	Emily	Davis	1995-07-25	emily.davis@example.com	+757688885
	5	Daniel	Brown	1987-03-14	daniel.brown@example.com	+757688886
	6	Sophia	Wilson	1993-06-22	sophia.wilson@example.com	+757688887
	7	Lucas	Taylor	1999-12-30	lucas.taylor@example.com	+757688888
	8	Olivia	Anderson	2001-09-11	olivia.anderson@example.com	+757688889
	9	Ethan	Moore	1992-02-05	ethan.moore@example.com	+757688822
	10	Ava	Martin	1989-11-18	ava.martin@example.com	+757688823

Figure 21 – Displaying the data stored in the customer table, showing unique CustID for each customer alongside their first name, last name, date of birth, email and formatted contact number. The sample data demonstrates the correct application of constraints, such as unique email addresses and contact numbers starting with a “+”.

Manufacturer Table

 10

	ManufacturerID	Name	Address	ContactNo
▶	1	CyclePros	123 Main Street, London	+447700900123
	2	TrailBlazers	456 Elm Avenue, Manchester	+447812345678
	3	SpeedRiders	789 Oak Lane, Birmingham	+447934567890
	4	BikeMasters	321 Pine Road, Leeds	+447945678901
	5	UrbanCycles	654 Cedar Avenue, Bristol	+447856789012
	6	AdventureGear	987 Spruce Boulevard, Glasgow	+447967890123
	7	VelocityWorks	159 Birch Drive, Edinburgh	+447978901234
	8	HighlandCycles	753 Maple Court, Inverness	+447989012345
	9	GearUp	951 Walnut Plaza, Cardiff	+447990123456
	10	RiderTech	852 Cherry Loop, Liverpool	+447901234567

Figure 22 - Listing details for each manufacturer, including the ManufacturerID, name, address, and contact number. The data confirms that the check constraint for the contact numbers has been applied correctly.



Category Table

	CategoryID	Name
▶	1	Mountain Bikes
	2	Road Bikes
	3	Accessories
	4	Electric Bikes
	5	Hybrid Bikes
	6	Kids Bikes
	7	Cruisers
	8	Fat Bikes
	9	Folding Bikes
	10	Gravel Bikes

Figure 23 – Showing the different categories available, such as “Mountain Bike” and “Accessories”. Each category is identified by a unique CategoryID, verifying that the category table correctly holds distinct values.

Product Table

	ProductID	Name	Description	Price	Quantity	ManufacturerID	CategoryID
▶	1	Trail Blazer 3000	High-performance mountain bike	1200.00	10	1	1
	2	Road Warrior Pro	Lightweight road bike	1500.00	5	2	2
	3	Cycling Helmet	Safety helmet for all riders	100.00	50	2	3
	4	Bike Lights	Front and rear LED bike lights	40.00	100	1	3
	5	Electric Commuter	Electric bike for city commuting	2200.00	7	3	4
	6	Hybrid Explorer	Versatile bike for road and trails	900.00	12	4	5
	7	Kid Cruiser	Safe and fun bike for kids	250.00	20	5	6
	8	Beach Rider	Comfortable cruiser for beach rides	500.00	8	6	7
	9	Fat Tire Beast	Heavy-duty fat tire bike	1700.00	6	7	8
	10	Compact Fold	Easy-to-store folding bike	800.00	15	8	9

Figure 24 – Displaying information for each product, including its name, description, price, and stock quantity. The ManufacturerID and CategoryID columns confirms that products are correctly linked to their respective manufacturers and categories. The data also reflects the enforcement of check constraints for positive prices and non-negative quantities.



Orders Table

	OrderID	CustID	ProductID	ProductQuantity	Date
▶	1	1	1	2	2024-12-01
	2	2	2	1	2024-12-02
	3	3	3	3	2024-12-03
	4	4	4	1	2024-12-04
	5	5	5	1	2024-12-05
	6	6	6	2	2024-12-06
	7	7	7	1	2024-12-07
	8	8	8	1	2024-12-08
	9	9	9	1	2024-12-09
	10	10	10	2	2024-12-10

Figure 25 – Recording customer purchases, showing the OrderID, CustID, ProductID, ProductQuantity, and the date of the order. The CustID and ProductID foreign keys demonstrate the connection between customers, products, and orders, ensuring referential integrity.

Invoice Table

	InvoiceID	OrderID	CustID	TotalPrice	PaymentStatus
▶	1	1	1	2400	Paid
	2	2	2	1500	Paid
	3	3	3	300	Pending
	4	4	4	40	Paid
	5	5	5	2200	Paid
	6	6	6	1800	Paid
	7	7	7	250	Pending
	8	8	8	500	Paid
	9	9	9	1700	Pending
	10	10	10	1600	Paid

Figure 26 – Presenting data from the invoice table, including InvoiceID, OrderID, CustID, TotalPrice, and PaymentStatus. The records show both “Paid” and “Pending” statuses, confirming that the sample data accurately represents different payment scenarios.

Review Table

	ReviewID	CustID	ProductID	Rating	ReviewText
▶	1	1	1	5	Fantastic bike, exceeded expectations!
	2	2	2	4	Great bike, but a bit pricey.
	3	4	4	5	Bike lights are super bright and long-lasting.
	4	5	5	5	The electric commuter bike is a game changer!
	5	6	6	4	Hybrid bike is versatile and sturdy.
	6	8	8	3	Comfortable, but not ideal for long rides.
	7	10	10	5	Folding bike is very convenient and compact.

Figure 27 – Showing customer feedback for purchased products, including the ReviewID, CustID, ProductID, Rating, and ReviewText. The ratings fall within the enforced range of 1 to 5, as required by the check constraint, and each review is correctly associated with valid customers and products through the foreign keys.



SQL Queries for Testing

Query 1



Fetching Customers Who Left Reviews:

```

223  -- 1) Fetch customers who left reviews.
224  •  SELECT
225      Customer.FName,
226      Customer.LName,
227      Review.Rating,
228      Review.ReviewText
229  FROM
230      Customer
231  INNER JOIN
232      Review ON Customer.CustID = Review.CustID;
233  -- This query works and brings back the desired output.

```

Figure 28 – Showing the SQL query used to fetch the first and last names of customers who left reviews, along with their rating and review text. The inner join between the customer and review tables ensures that only customers who have left reviews are displayed.

	FName	LName	Rating	ReviewText
▶	John	Doe	5	Fantastic bike, exceeded expectations!
	Jane	Smith	4	Great bike, but a bit pricey.
	Emily	Davis	5	Bike lights are super bright and long-lasting.
	Daniel	Brown	5	The electric commuter bike is a game changer!
	Sophia	Wilson	4	Hybrid bike is versatile and sturdy.
	Olivia	Anderson	3	Comfortable, but not ideal for long rides.
	Ava	Martin	5	Folding bike is very convenient and compact.

Figure 29 – Displaying the output table generated by the SQL query in Figure 28. The table shows customer names, ratings, and review feedback, confirming that the query correctly retrieved the relevant data from the customer and review tables.



Query 2

Calculating Total Sales Per Product:

```

235  -- 2) Calculating total sales per product
236  •  SELECT
237      Product.Name AS ProductName,
238      SUM(Orders.ProductQuantity) AS TotalQuantitySold, -- Using the 'AS' changes the
239                                                         -- column name to make it more
240                                                         -- readable.
241      SUM(Orders.ProductQuantity * Product.Price) AS TotalRevenue
242  FROM
243      Orders
244  INNER JOIN
245      Product ON Orders.ProductID = Product.ProductID
246  GROUP BY
247      Product.Name;
248  -- This query worked adding up the items sold and grouping similar items and combining
249  -- the overall revenue.

```

Figure 30 – Presenting the SQL query for calculating total sales per product. The query sums the ProductQuantity and calculates the total revenue by multiplying the quantity sold by the product price. The inner join links the orders and product tables by ProductID, and the GROUP BY clause groups the results by product name.

	ProductName	TotalQuantitySold	TotalRevenue
▶	Trail Blazer 3000	2	2400.00
	Road Warrior Pro	1	1500.00
	Cycling Helmet	3	300.00
	Bike Lights	1	40.00
	Electric Commuter	1	2200.00
	Hybrid Explorer	2	1800.00
	Kid Cruiser	1	250.00
	Beach Rider	1	500.00
	Fat Tire Beast	1	1700.00
	Compact Fold	2	1600.00

Figure 31 – Showing the output table generated by the query in Figure 30. It lists each product's name, the total quantity sold, and the total revenue generated. The results confirm that the query correctly summarized the sales data.



Query 3

Fetching Customers with Pending Payments:

```
251 -- 3) Fetching customers with pending payments
252 • SELECT
253     Customer.FName,
254     Customer.LName,
255     Invoice.TotalPrice,
256     Invoice.PaymentStatus
257 FROM
258     Customer
259 INNER JOIN
260     Invoice ON Customer.CustID = Invoice.CustID
261 WHERE
262     Invoice.PaymentStatus = 'Pending';
263 -- This query worked bring back the three customers who's payments are currently pending.
```

Figure 32 – Showing the SQL query used to fetch customers with pending payments. The query joins the customer and invoice tables by CustID and filters for invoices on PaymentStatus of “Pending”.

	FName	LName	TotalPrice	PaymentStatus
▶	Michael	Johnson	300	Pending
	Lucas	Taylor	250	Pending
	Ethan	Moore	1700	Pending

Figure 33 – Representing the output table for the query in Figure 32. It lists the names of customers with pending payments, the total amount owed, and the payment status. The results confirm that the query successfully retrieved customers with incomplete payments.



Query 4

Finding Products with a Reviewed Rating of 4 or Higher:

```

265  -- 4) Finding products with a reviewed rating of 4 or higher.
266  •  SELECT
267      Product.Name AS ProductName,
268      Review.Rating,
269      Review.ReviewText
270  FROM
271      Product
272  INNER JOIN
273      Review ON Product.ProductID = Review.ProductID
274  WHERE
275      Review.Rating >= 4; -- >= more than or equal too
276  -- This query worked showing review ratings of 4 or higher.

```

Figure 34 – Displaying the SQL query for finding products with a review rating of 4 or higher. The query joins the product and review tables by ProductID and filters for reviews where the rating is 4 or higher.

	ProductName	Rating	ReviewText
▶	Trail Blazer 3000	5	Fantastic bike, exceeded expectations!
	Road Warrior Pro	4	Great bike, but a bit pricey.
	Bike Lights	5	Bike lights are super bright and long-lasting.
	Electric Commuter	5	The electric commuter bike is a game changer!
	Hybrid Explorer	4	Hybrid bike is versatile and sturdy.
	Compact Fold	5	Folding bike is very convenient and compact.

Figure 35 – Showing the output table for the query in Figure 34. It lists the product names, ratings, and review comments for products that receive high ratings. The output confirms that the query correctly filtered and retrieved the expected data.



Query 5

Fetching All Products and Their Manufacturer Details:

```

278 -- 5) Fetching all products and their manufacturer details.
279 • SELECT
280     Product.Name AS ProductName,
281     Manufacturer.Name AS ManufacturerName,
282     Manufacturer.Address,
283     Manufacturer.ContactNo
284 FROM
285     Product
286 INNER JOIN
287     Manufacturer ON Product.ManufacturerID = Manufacturer.ManufacturerID;
288 -- This query worked showing a list of products along with their manufacturer details.

```

Figure 36 – Showing the SQL query used to fetch all products along with their manufacturer details. The query joins the product and manufacturer tables by ManufacturerID to display manufacturer name, address, and contact number alongside each product.

	ProductName	ManufacturerName	Address	ContactNo
▶	Trail Blazer 3000	CyclePros	123 Main Street, London	+447700900123
	Road Warrior Pro	TrailBlazers	456 Elm Avenue, Manchester	+447812345678
	Cycling Helmet	TrailBlazers	456 Elm Avenue, Manchester	+447812345678
	Bike Lights	CyclePros	123 Main Street, London	+447700900123
	Electric Commuter	SpeedRiders	789 Oak Lane, Birmingham	+447934567890
	Hybrid Explorer	BikeMasters	321 Pine Road, Leeds	+447945678901
	Kid Cruiser	UrbanCycles	654 Cedar Avenue, Bristol	+447856789012
	Beach Rider	AdventureGear	987 Spruce Boulevard, Glasgow	+447967890123
	Fat Tire Beast	VelocityWorks	159 Birch Drive, Edinburgh	+447978901234
	Compact Fold	HighlandCycles	753 Maple Court, Inverness	+447989012345

Figure 37 – Presenting the output table for the query in Figure 36. It shows each product's name the associated manufacturer's name, address, and contact number. The output verifies that the query correctly linked and retrieved data from both tables.



Transaction Management

Commit

```
290 -- Transaction Management
291 • START TRANSACTION; -- This begins the transaction
```

Figure 38 – Showing the START TRANSACTION command, which initiates a new transaction. All subsequent operations will either be committed together or rolled back, ensuring data consistency during the transaction process.

```
293 -- Adding a new order into the database
294 • INSERT INTO Orders (OrderID, CustID, ProductID, ProductQuantity, Date)
295 VALUES (11, 3, 2, 1, '2024-12-11');
```

Figure 39 – Showing the INSERT INTO Orders command, which adds a new order record to the orders table. The values inserted correspond to the columns in the orders table along with a quantity of 1, with the purchase date set to 2024-12-11.

When running the SELECT * FROM Orders table the following output is generated, showing that the order has been added.

	OrderID	CustID	ProductID	ProductQuantity	Date
▶	1	1	1	2	2024-12-01
	2	2	2	1	2024-12-02
	3	3	3	3	2024-12-03
	4	4	4	1	2024-12-04
	5	5	5	1	2024-12-05
	6	6	6	2	2024-12-06
	7	7	7	1	2024-12-07
	8	8	8	1	2024-12-08
	9	9	9	1	2024-12-09
	10	10	10	2	2024-12-10
	11	3	2	1	2024-12-11

13

Figure 40 – Displaying the output of the SELECT * FROM Orders query, confirming that the new order has been successfully added to the Orders table. The new row for OrderID 11 is shown at the bottom of the result set.

```
296 -- Checking to see if the product quantity
297 • SELECT Quantity FROM Product WHERE ProductID = 2;
```

Figure 41 – Showing the SQL query, which retrieves the current stock quantity for the product with ProductID 2. This step checks whether there is enough stock available to fulfil the order.

	Quantity
▶	5

Figure 42 – Showing the result of the SELECT query, indicating that the current stock quantity for ProductID 2 is 5. Since this value is greater than 0, the transaction can proceed without issues.



```

298  -- As no concerns can commit
299  • UPDATE Product SET Quantity = Quantity - 1 WHERE ProductID = 2;
300  • COMMIT;

```

Figure 43 – Showing the UPDATE Product command, which reduces the stock quantity for ProductID 2 by 1. The commit command saves the changes made during the transaction, confirming that the new order is finalized, and the stock has been updated.

When the command is then run to see if the ProductID 2 has been reduced by one in the products table it can be seen as reduced from 5 items down to four items remaining.

	ProductID	Name	Description	Price	Quantity	ManufacturerID	CategoryID
▶	1	Trail Blazer 3000	High-performance mountain bike	1200.00	10	1	1
	2	Road Warrior Pro	Lightweight road bike	1500.00	4	2	2
	3	Cycling Helmet	Safety helmet for all riders	100.00	50	2	3
	4	Bike Lights	Front and rear LED bike lights	40.00	100	1	3
	5	Electric Commuter	Electric bike for city commuting	2200.00	7	3	4
	6	Hybrid Explorer	Versatile bike for road and trails	900.00	12	4	5
	7	Kid Cruiser	Safe and fun bike for kids	250.00	20	5	6
	8	Beach Rider	Comfortable cruiser for beach rides	500.00	8	6	7
	9	Fat Tire Beast	Heavy-duty fat tire bike	1700.00	6	7	8
	10	Compact Fold	Easy-to-store folding bike	800.00	15	8	9

Figure 44 - Displaying the updated product table after the commit command has been executed. The quantity for ProductID 2 (Road Warrior Pro) has been reduced from 5 to 4, confirming that the transaction successfully updated the stock quantity as expected after the new order was placed.



Rollback

```
302 -- Transaction Management Rollback
303 • START TRANSACTION; -- This begins the transaction
```

Figure 45 – Showing the start of the rollback transaction process with the START TRANSACTION command. This begins the transaction.

```
305 -- Adding a new order into the database
306 • INSERT INTO Orders (OrderID, CustID, ProductID, ProductQuantity, Date)
307 VALUES (12, 4, 2, 10, '2024-12-12');
```

Figure 46 – Displaying the INSERT INTO Orders statement, which attempts to insert as order requesting 10 units of ProductID 2. This is used to demonstrate what happens when the requested quantity exceeds available stock.

	OrderID	CustID	ProductID	ProductQuantity	Date
▶	1	1	1	2	2024-12-01
	2	2	2	1	2024-12-02
	3	3	3	3	2024-12-03
	4	4	4	1	2024-12-04
	5	5	5	1	2024-12-05
	6	6	6	2	2024-12-06
	7	7	7	1	2024-12-07
	8	8	8	1	2024-12-08
	9	9	9	1	2024-12-09
	10	10	10	2	2024-12-10
	11	3	2	1	2024-12-11
	12	4	2	10	2024-12-12

Figure 47 – Showing the output of the SELECT * FROM Orders query, confirming that the attempted order with OrderID 12 was temporarily added during the transaction process before checking stock availability.

```
309 -- Check the current stock
310 • SELECT Quantity FROM Product WHERE ProductID = 2;
```

Figure 48 – Showing the SQL code to view the Quantity of the requested item. This ensures the system verifies whether an order can be fulfilled.

	Quantity
▶	4

Figure 49 – Displaying the result of the stock check, indicating that there are only four units available for ProductID 2. Since the order requested 10 units, this triggers the rollback process.

As the customer wished to purchase 10 items from ProductID 2 and there are only four items available. The command rollback will now be used.

```
312 • ROLLBACK; -- quantity is insufficient, performed a rollback
```

Figure 50 – Showing the rollback command being executed due to insufficient stock. This will now cancel the entire transaction, ensuring no changes are saved to the database for this failed order.



To ensure to rollback has taken place the command to view the output of the orders table is used
SELECT * FROM Orders.

	OrderID	CustID	ProductID	ProductQuantity	Date
▶	1	1	1	2	2024-12-01
	2	2	2	1	2024-12-02
	3	3	3	3	2024-12-03
	4	4	4	1	2024-12-04
	5	5	5	1	2024-12-05
	6	6	6	2	2024-12-06
	7	7	7	1	2024-12-07
	8	8	8	1	2024-12-08
	9	9	9	1	2024-12-09
	10	10	10	2	2024-12-10
	11	3	2	1	2024-12-11
*	NULL	NULL	NULL	NULL	NULL

Figure 51 – Showing the result of the SELECT * FROM Orders query after the rollback. The attempted order with OrderID 12 has been removed, confirming that the rollback successfully undid the transaction, preserving the integrity of the orders table.



Database Performance Indexes

To enhance the performance of the database, indexes have been added to key columns that are frequently used in queries [5][6].

```
302 -- Database Performance Indexes
303 • CREATE INDEX idx_customer_email ON Customer (CustEmail); -- Index for faster searches by email
304 • CREATE INDEX idx_orders_custid ON Orders (CustID); -- Index for faster joins by CustID
305 • CREATE INDEX idx_orders_productid ON Orders (ProductID); -- Index for faster joins by ProductID
306 • CREATE INDEX idx_review_productid ON Review (ProductID); -- Index for faster review lookups by Product
307 • CREATE INDEX idx_review_custid ON Review (CustID); -- Index for faster review lookups by customer
308 • CREATE INDEX idx_invoice_orderid ON Invoice (OrderID); -- Index for quicker invoice retrieval by OrderID
```

Figure 52 – Six indexes have been added to the database to make queries on commonly search criteria faster.

Indexes act as shortcuts that help the database quickly locate records without scanning the entire table, improving query efficiency especially for SELECT statements with WHERE, JOIN, ORDER BY, or GROUP BY clauses [7].

14

For example, searches for orders by CustID or reviews by ProductID run faster due to indexes on these columns. By indexing commonly queried fields like CustEmail in the customer table and ProductID in the orders and review tables, the database handles lookups and joins more efficiently, even as the dataset grows.

15



Report Conclusion

In conclusion, this report outlines the design, creation, and testing of a relational database for managing orders, customers, products, and reviews. A structured approach ensured that tables, relationships, and constraints-maintained data integrity while supporting key operations such as data insertion, updates, and queries.

Transaction management handled successful data changes (commit) and prevented invalid operations (rollback), ensuring consistency. Performance tuning, including indexing, improved query efficiency for large datasets.

Overall, the database design meets real-world requirements by supporting reliable data operations, efficient retrieval, error prevention, and performance optimization.



References

- [1] Wikipedia Contributors, "Entity–relationship model," *Wikipedia*, May 13, 2019.
[/https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model](https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model) (accessed 15/12/2024)
- [2] T. J. Teorey, *Database modeling and design : logical design*, 5th ed. Amsterdam ; Morgan Kaufmann Publishers, 2011. (accessed 15/12/2024).
- [3] T. M. Connolly and C. E. Begg, *Database Systems : a practical approach to design, implementation, and management*, Sixth, Global edition. Boston: Pearson, 2015. (accessed 15/12/2024)
- [4] J. L. Harrington and J. L. Harrington, *Relational database design and implementation : clearly explained*, 3rd ed. Amsterdam ; Morgan Kaufmann/Elsevier, 2009. (Accessed 12/12/2024) (Chapter 6 third normal form)
- [5] Wikipedia Contributors, "Database index," *Wikipedia*, Nov. 15, 2019.
https://en.wikipedia.org/wiki/Database_index (accessed 12/12/2024)
- [6] GeeksforGeeks, "Indexing in Databases | Set 1 - GeeksforGeeks," *GeeksforGeeks*, Jul. 15, 2016.
<https://www.geeksforgeeks.org/indexing-in-databases-set-1/> (accessed 12/12/2024)
- [7] J. D. Ullman and J. Widom, *A first course in database systems*, Third edition. Harlow, England: Pearson Education Limited, 2014. (accessed 12/12/2024) (chapter 8 Vies & Indexes)





2311568 - Database Assignment Report

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

87 /100

PAGE 1

PAGE 2



Comment 1 | Transferable Skills

Table of Contents demonstrates professionalism and attention to detail



Comment 2 | Transferable Skills

Numbering helps readers quickly locate sections and sub-sections based on their position in the document, creating a logical structure

PAGE 3



Comment 3 | Transferable Skills

The introduction provides a brief explanation of the business problem, outlining AdventureWorks Cycles

PAGE 4



Comment 4 | Knowledge&Understanding Skills

The ERD includes all key entities which are essential for the database design

PAGE 5



Comment 5 | Knowledge&Understanding Skills

The ERD adheres to 3rd Normal Form, but the explanation should have started from 1NF to show the normalization process step by step

**Comment 6** | Cognitive &Intellectual Skills

screenshots are neatly organized, help maintain the report's readability and professionalism, making it easy for the reader to follow your process.

**Comment 7** | Cognitive &Intellectual Skills

Some descriptions could be shortened since they are already in the code comments. This would help stay within the word limit. Please condense the content while keeping the key points clear

**Comment 8** | Practical Application Skills

evidence of constraint checking is missing

**Comment 9** | Cognitive &Intellectual Skills

screenshots focus on key points however the output window after inserting(select command),checking constraints.... is missing

**Comment 10** | Cognitive &Intellectual Skills

could have been aligned with the SQL code for better clarity and easier navigation

**Comment 11** | Practical Application Skills

The SQL statements used to create tables, implement constraints, and insert data are clear and well-structured



Comment 12 | Practical Application Skills

thoughtful approach of including Queries

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21



Comment 13 | Practical Application Skills

rollback and commit example is based on a real-world scenario, it shows a practical understanding

PAGE 22

PAGE 23

PAGE 24

PAGE 25



Comment 14 | Practical Application Skills

Indexing can significantly improve the speed of queries, especially on large datasets.



Comment 15 | Practical Application Skills

performance testing before and after indexing is not thoroughly conducted (e.g., no comparison of query execution times)

PAGE 26



Comment 16 | Transferable Skills

conclusion provides a clear sense of closure to the report, summarizing the work done however doesn't address any limitations or areas for future improvement

PAGE 27



Comment 17 | Literature Skills

literature review effectively links theoretical concepts to the practical implementation discussed in the report.



Comment 18 | Literature Skills

Expand the scope of the literature review to include a wider range of relevant topics such as performance tuning, scalability, and best practices in database administration, ensuring all aspects of the project are covered.

GRADING FORM: DAD_2024

NATE FOX

87

LITERATURE SKILLS



8

KNOWLEDGE&UNDERSTANDING SKILLS



16.5

COGNITIVE &INTELLECTUAL SKILLS



18

PRACTICAL APPLICATION SKILLS



26

TRANSFERABLE SKILLS



18.5