Report

Conceptual/ E-R Model

The conceptual model follows the typical customer journey through the e-commerce platform, introducing entities and relationships at each stage.

Customers

Each customer has a unique id, email, name, date of birth, and optional phone number (customers entity). Emails are unique to prevent duplicate accounts.

Products and sellers

Products have attributes like name, description, brand, colour, dimensions, weight, price, warranty, category, stock, and a reference to a seller. Each product belongs to one seller, while a seller can offer many products (sellers entity with seller_code, name, address, tax ID, rating).

Discounts

A product may have one discount (discounts entity), which is optional and stores the discount amount and validity.

Basket

Each customer has one active basket. The basket_items table links customers and products in a many-to-many relationship, with a quantity attribute.

Payments and cards

Customers pay via vouchers or cards. Cards are stored in a cards table (one-to-many from customer). Payments records each transaction with amount, method, status, date, and associated order.

Orders

Orders belong to customers and include order date, payment status, and total amount. Products in orders are modelled via the many-to-many order_items table (with quantity). Each order may have a single delivery (deliveries entity).

Reviews and returns

Customers can leave reviews (reviews) and request returns (returns). Returns involve multiple products through return_items (quantity included). Both reviews and returns are many-to-many relationships with products.

Assumptions

- Surrogate integer keys used for all strong entities
- Baskets are transient; order items must exist, but products may not yet be ordered
- Each delivery corresponds to one order

This design captures the main e-commerce workflow while maintaining clear cardinalities, optional participation, and logical relationships.

Relational Model

The relational model was derived by looking at the conceptual model and creating tables, foreign keys and appropriate constrains.

Each strong entity in the conceptual model (such as `customers`, `products`, `sellers`, `orders`, `payments`, and `returns`) has been mapped to a base table with a surrogate integer primary key (`id`). Attributes from the E-R model became columns in their respective tables, maintaining the same data semantics.

Relationships

Customers:

- Customers can may have many cards or vouchers that can pay with $(1 : 0 \dots N)$ with `cards` / `vouchers`
- Customers can place many orders, but customers can have no orders giving $(1 : 0 \dots N)$ with `orders`
- Customers can have items in a basket, but a product can appear in multiple customers basket, giving a $(N : M)$ relationship with `products`

Products:

- There is exactly 1 seller for each product, but each seller can sell multiple products $(N : 1)$ with `sellers`
- Products can have multiple reviews, but not all products will have reviews $(1 : 0 \dots N)$ with `reviews`

- Multiple products can be ordered or returned and multiple orders/returns can have multiple so there is a $(N : M)$ with `orders` / `returns`. This is created using the `order_items` and the `return_items` tables
- Products can have none or one promotion. Giving a $(1 : 0..1)$ with `promotions`

Orders:

- Each order can have a delivery, giving a $(1 : 0..1)$ with `deliveries`
- Each order can also have a return, giving a $(1 : 0..1)$ with `returns`
- Each order also has exactly one payments, giving a $(1 : 1)$ with `payments`

Payments:

- Each payment is associated with ether a card or a voucher, giving a $(1 : 0..1)$ with `cards` and `vouchers`

Views

The client might want to know which products are in low stock, we assume that a stock count less than 10 is low, to make this easily accessible I have crated a view that displays the product id, product name, product price, product stock price, seller name, and seller email (perhaps to message them about low stock counts)

Data Sources

The simplest way of creating data would be to provide an LLM with our database schema and prompt it to produce synthetic data, an LLM would create a small number of rows for each table and then its up to us double check this and ensure that all the constraints are met and the foreign key attributes are correct. This is what I initially used to check my data. (Ref 2)

Using python and faker

We can use the python faker library to along side probabilistic distributions to simulate realistic marketplace behaviour at a larger scale than with LLMs. It assigns customers, sellers, products, orders, and reviews with attributes drawn from weighted and normal distributions rather than uniform randomness. Product prices follow a normal distribution centred around typical price ranges, while product categories are chosen based on their relative popularity. Customer activity levels determine how many orders they place, with most making only a few purchases. Reviews are biased toward higher ratings, and payment or return

statuses occur according to realistic probabilities. Quantities in orders and baskets are more likely to be small, and seller ratings cluster around generally positive averages. This approach produces coherent, varied, and lifelike data that closely resembles actual e-commerce patterns.

SQL Programming

5.1. Customers Order History

This question involves getting all the customers and their orders. This question involves gathering data across different tables and gathering together, and so the relational nature of our database can be used here. Each customer may have several orders and each one may have several order items (products). Each order is also linked with a payment, and in this case we want only "paid" or "pending" payments.

The screenshot below shows the first 10 rows of the output

name	email	order_id	order_date	amount_total	product_name	product_price	payment_status
Kevin French	dean38@example.org	1389108910	2023-12-10	1338.0	Julie Green	223.0	pending
Kevin French	dean38@example.org	1389108910	2023-12-10	1338.0	Julie Green	223.0	pending
Kevin French	dean38@example.org	1389108910	2023-12-10	1338.0	Julie Green	223.0	pending
Kevin French	dean38@example.org	1389108910	2023-12-10	1338.0	Julie Green	223.0	pending
Kevin French	dean38@example.org	1389108910	2023-12-10	1338.0	Julie Green	223.0	pending
Kevin French	dean38@example.org	1389108910	2023-12-10	1338.0	Julie Green	223.0	pending
Kevin French	dean38@example.org	1389108910	2023-12-10	1338.0	Julie Green	223.0	pending
Kevin French	dean38@example.org	1389108910	2023-12-10	1338.0	Julie Green	223.0	pending
Kevin French	dean38@example.org	1389108910	2023-12-10	1338.0	Julie Green	223.0	pending
Kevin French	dean38@example.org	1389108910	2023-12-10	1338.0	Julie Green	223.0	pending

5.2. Basket Analysis

Since we have chosen that each customer only has one basket at any given time, this question amount to performing joins on the `basket_items` table. We retrieve other attributes from the joined tables.

customer_id	customer_name	customer_email	voucher_amount	product_name
5887023112	Jacqueline Fitzpatrick	schroederdonna@example.net		Julie Jones
9814819341	Mrs. Valerie Jordan	anna98@example.net		Brenda Gonzalez
8546228764	Sherri Case	ksalazar@example.org		Charles Munoz
8630772766	Elizabeth Lee	tina56@example.com		Steven Rodriguez
2083527200	Robert Whitehead	rodriguezrick@example.net		Matthew Erickson
6681594913	Michael Page	robert45@example.net	90.0	Alison Gonzalez
9947015205	Jocelyn Hodges	michelle80@example.org		Mathew Pena
6872074235	Laura Norris	iwashington@example.net		Christopher Gibson
9160681515	Michelle Brown	carlgonzalez@example.com		Steven Rodriguez
8042268718	Lauren Johnson	michellecox@example.com		Sara Stokes

5.3. Top products by revenue

This question involves 3 main steps. Firstly calculating revenues of each product in a category, then ranking them and then getting the top 3 ones. I have used window functions to calculate revenues and also to add rankings to each of the

products revenue. From then it involved filtering out the top 3 products in each category.

From the synthetic data, the products are given as (human) names.

product_category	product_name	revenue
beauty	Elizabeth Snyder	1599.0
beauty	Deborah Carlson	1155.0
beauty	Amber Miller	1056.0
books	Justin Wagner	1089.0
books	Joshua Anderson	282.0
electronics	Richard Cortez	2820.0
electronics	Kathleen Perez	2250.0
electronics	Kevin Matthews	1260.0
fashion	Thomas Mitchell	1755.0
fashion	Mary Avila	1410.0
fashion	Mr. Ryan Nelson	1377.0
home	Jennifer Holt	2132.0
home	Tina Harris	1755.0
home	Michele Gordon	1716.0
sports	Kathy Lozano	2660.0
sports	Emily Lopez	2268.0
sports	Pamela Jones	1342.0
toys	John Duffy	1250.0
toys	Jacob Castaneda	550.0
toys	Erika Nichols	72.0

5.4. Sales Trends

Similar to the previous question, we can split this question up into two parts. First involves getting the monthly sales, to do this I need to calculate how many items have been ordered and subtract them with the amount that have been returned. I have a used a the `COALSE()` function here to default the return quantity to 0 if no items have been returned from that ordered. Using the `SUM()` window function I can sum up each of the sales. Then to group by month I have used that all my dates that have been inserted into my DB are in ISO 8601 standard which allows me to use the `STRFTIME()` formatting function to group by year-month. The next step would be to calculate the percentage change, this would mean that we have to look at the sales from the row above after ordering by `order_month`. Here we can use the `LAG()` window function, this avoids the need of a self-join.

Since the synthetic data of prices and orders and returns is chosen uniformly, we can see that the growth rate hovers around 0. But if we used real world data we

would be able to see more robust trends.

order_month	product_count	sales	prev_month_sales	growth_rate
2023-11	11	30921.0		
2023-12	24	39428.0	30921.0	0.275120468290159
2024-01	12	36521.0	39428.0	-0.0737293294105712
2024-02	14	29518.0	36521.0	-0.191752690233017
2024-03	17	44765.0	29518.0	0.516532285385189
2024-04	22	64852.0	44765.0	0.448721099072936
2024-05	14	37306.0	64852.0	-0.424751742428915
2024-06	15	34047.0	37306.0	-0.0873586018334852
2024-07	10	27097.0	34047.0	-0.204129585572885
2024-08	18	47945.0	27097.0	0.769384064656604
2024-09	9	22499.0	47945.0	-0.530733131713422
2024-10	13	29567.0	22499.0	0.314147295435353
2024-11	16	33496.0	29567.0	0.132884634897014
2024-12	15	39778.0	33496.0	0.187544781466444
2025-01	16	27430.0	39778.0	-0.310422846799739
2025-02	10	26852.0	27430.0	-0.0210718191760846
2025-03	19	37889.0	26852.0	0.411030835691941
2025-04	20	30569.0	37889.0	-0.193195914381483
2025-05	17	42447.0	30569.0	0.388563577480454
2025-06	16	43574.0	42447.0	0.0265507574151295
2025-07	15	23393.0	43574.0	-0.463143158764401
2025-08	13	35286.0	23393.0	0.508399948702603
2025-09	17	29893.0	35286.0	-0.152836819135068
2025-10	13	31084.0	29893.0	0.0398421035024922
2025-11	3	2959.0	31084.0	-0.904806331231502

6. DB consistency

I created triggers for situations `(i)` and `(iv)` .

For situation `(i)` this involved checking if the quantity of each product in the new row inserted into the `order_items` table, i.e. the quantity of products being purchased, is greater than the available stock count of that product. For situation `(iv)` this involves updating product `stock_count` after there was an order. We simply subtract the current stock count by the ordered amount.

The screenshot below shows both of the triggers in action. We select a random product id and then we add an item to an order with quantity less that of the stock count and see no error and that the stock count has decreased. If we try to insert another row into `order_items` which has a higher amount of quantity than `stock_count` we see that we get an error saying `Insufficient stock` .

```
sqlite> select id, stock_count from products where id=12894;
+-------+-------------+
|  id   | stock_count |
+-------+-------------+
| 12894 | 44          |
+-------+-------------+
sqlite> insert into order_items values(1, 12894, 10);
sqlite> select id, stock_count from products where id=12894;
+-------+-------------+
|  id   | stock_count |
+-------+-------------+
| 12894 | 34          |
+-------+-------------+
sqlite> insert into order_items values(1, 12894, 35);
Runtime error: Insufficient stock (19)
```

For consistency, I also wrote a trigger to populate the `amount_total` from the `order_items` basket, which calculates the total. This helps for question 5.3 and 5.4.

References

1. main chat log: https://chatgpt.com/share/690ea647-7b1c-8012-baff-060d701ab67e
2. synthetic data chat log: https://chatgpt.com/share/690ea684-ce10-8012-8edb-2990449faeaf