Normalized Table (Up to 3NF)

Order Table (PK = order_id):

order_id	**customer_id**	**order_date**	**total**
001	1	2023-11-01	1000
002	2	2023-11-02	1000
003	3	2024-11-02	50
004	4	2024-11-03	10

Customer Table (PK = customer_id):

customer_id	**customer_name**	**customer_email**
1	Jean Doe	jsmith@lmail.com
2	Rebecca Yeboah	ryeb@lmail.com
3	Jean Doe	jdoe@lmail.com
4	Bertina Ayuure	bayuure@lmail.com

Customer Address Table (PK = address_id):

address_id	**customer_id**	**country**	**city**
1	1	Ghana	Accra
2	2	Rwanda	Kigali
3	3	Ghana	Accra
4	4	German	Bonn

Product Table (PK = product_id):

product_id	**product_name**	**category**	**price**	**Supplier_i**
101	Laptop	Electronics	1000	1
102	Phone	Electronics	500	2
103	Mouse	Electronics	25	3
104	Keyboard	Accessories	10	4

Supplier Table (PK = supplier_id):

supplier_id	**supplier_name**
1	CompuGhana
2	RoboTech
3	RapasatTech
4	T-Shop

Order_Item Table:

order_item_id	**order_id**	**customer_id**	**quantity**	**total**
1	001	1	1	1000
2	002	2	2	1000
3	003	3	2	50
4	004	4	1	10
PK + FK				

<u>Reason for design choices for primary and foreign keys</u>

In each table, I named the primary key with the table + _id. I choose not to make a primary key a composite one for simplicity and performance's sake. It simplifies the database schema, improves performance, and makes it easier to establish relationships between tables, while preserving the integrity of the original data. Using a surrogate key as I did in my design is also a widely used and recommended practice.

The relationship between orders and products is many to many relationship and relational databases do not directly support this kind of relationship. That is why I used intermediate/junction table (also called associative entity or table) called Order_Item Table to connect products and orders to reduce data redundancy and improve data integration.

How normalization helps reduce redundancy and improve data integrity

Normalization is the process of organizing data in a database to minimize redundancy and improve data integrity.

By dividing data into smaller, related tables and ensuring that the relationships between them are logical and consistent, normalization helps prevent unnecessary duplication of data and ensures that the database accurately reflects the real-world relationships it is meant to model.

It helps create a clean, structured, and efficient database, which simplifies management, reduces errors, and ensures data consistency across the system.

Potential trade-offs and when denormalization might be necessary

Normalization is the best approach to maintain data integrity, reducing redundancy, and ensuring consistency in most transactional database systems. However, denormalization may be necessary in some situations where trade-offs might be involved.

Denormalization refers to the process of combining tables, reintroducing redundancy, and reducing the level of normalization for performance reasons. It is the opposite of normalization, which aims to reduce redundancy and ensure data integrity.

Trade-offs of normalization

While normalization reduces redundancy and helps maintain data integrity, it can make the database slower when:

1. **Queries become complex:** Normalized data often requires joining multiple tables, which can slow down performance.
2. **Increased disk access:** Data spread across many tables means more dis I/O, leading to slower reads.
3. **Complex queries:** Joins and relationships across multiple tables can make queries harder to write and maintain.

When denormalization might be necessary

Denormalization can improve performance in certain cases, such as:

1. **When read performance is critical:** If your application reads data more often than it writes it (e.g., reporting, analytics), denormalization cand reduce the need for complex joins and speed up queries.
2. **For large, complex queries:** When you frequently need to retrieve large datasets that involve many joins, combining data into fewer tables can make queries faster.
3. **In data warehouse and OLAP:** Denormalization is common in data warehouses where large datasets need to be aggregated quickly for reporting and analysis.

The trade-offs of denormalization

While denormalization improves performance for certain queries, it comes with some downsides:

1. **More storage:** Redundant data means more space is required to store it.
2. **Data inconsistency risks:** Redundant data can become inconsistent if one part of the data is updated, and another isn't.
3. **Slower writes:** When data changes, it might need to be updated in multiple places, making writes (inserts, updates, deletes) slower.
4. **Harder maintenance:** Keeping the data consistent across multiple copies can add complexity.

When to avoid denormalization

- **Small to medium databases:** If your system isn't handling massive amounts pf data, normalization is typically sufficient.
- **If data integrity is more important:** When you need to ensure that your data is always consistent and up-to-date, normalization is a safer choice.