**Question 4**

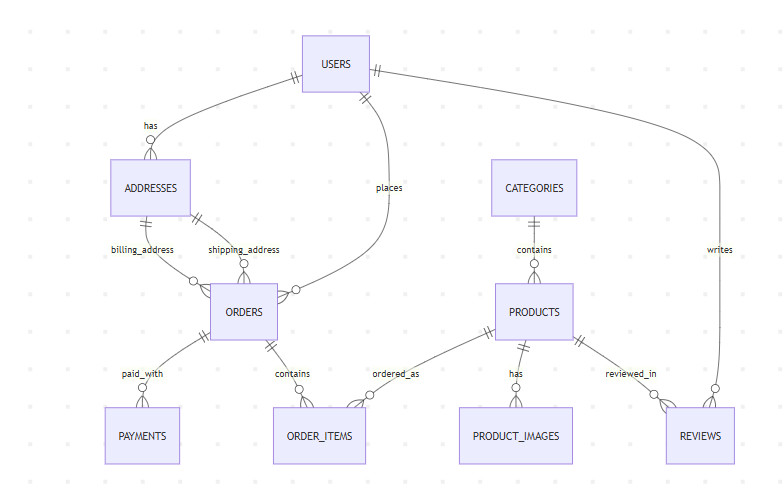
**Database design for E-commerce**

**Design an Oracle database schema for an e-commerce website ensuring data integrity, scalability and efficient query performance.**

For any **E-commerce application**, the core details we handle and store include:

1. **Authentication & User Details** – Stores customer and admin information, including login credentials, profiles, addresses, ensuring secure access management.
2. **Product Details** – Maintains product catalogs with attributes like name, category, price, description, and images to facilitate browsing and purchasing.
3. **Order Details** – Tracks customer orders, including order status, timestamps, and associated products, ensuring seamless order management.
4. **Inventory Details** – Manages stock levels, supplier details, and warehouse locations to prevent overselling and optimize supply chain operations.
5. **Payment Details** – Records transaction history, payment methods, and statuses, ensuring secure and seamless financial processing.
6. **Feedback & Recommendations** – Stores product reviews, ratings, and personalized recommendations to enhance user experience and boost sales.

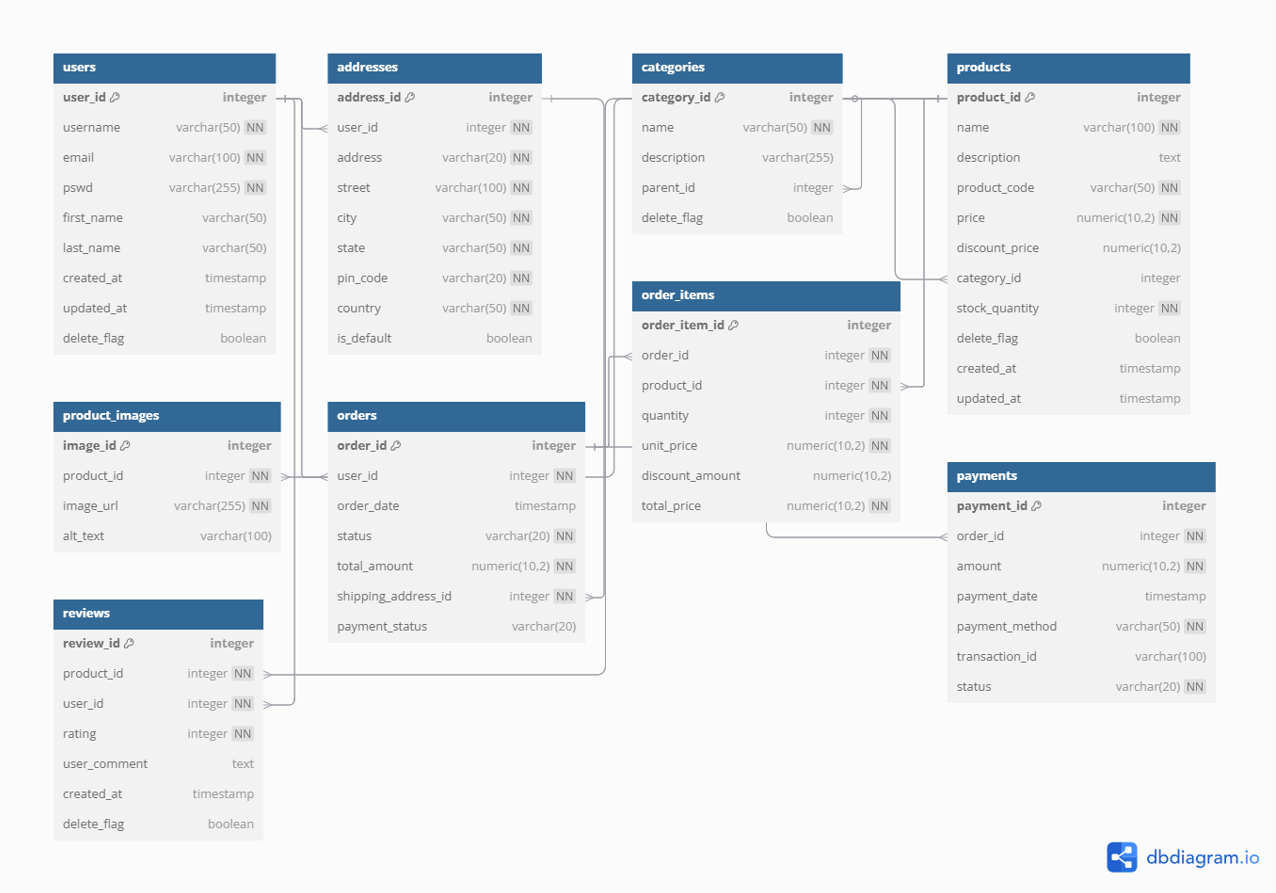
**High-level Flow**



**High level Business Logic:**

1. Customers (Users) place Orders containing Products
2. Products are organized in Categories and have multiple Images
3. Customers can review Products they've purchased
4. Address management supports shipping/billing needs

**ER Diagram**



**Core Tables:**

1. **Users**
2. Stores customer account details
3. Key fields are  user\_id, email, password, name fields
4. Relates to: Addresses, Orders, Reviews
5. **Addresses**
   1. Stores shipping/billing addresses
   2. Key fields: address\_id, user\_id (FK), address components
   3. Relates to: Users (many-to-one) and Orders
6. **Categories**
   1. Products are classification into their respective categories
   2. Key fields: category\_id, name, parent\_id (for subcategories)
   3. Relates to: Products (one-to-many)
7. **Products**
   1. Core product catalog
   2. Key fields: product\_id, name, produce\_code, pricing, category\_id (FK)
   3. Relates to: Categories, Product\_Images, Order\_Items, Reviews
8. **Product\_Images**
   1. Table to store the products images
   2. Key fields: image\_id, product\_id (FK), image\_url, is\_primary
   3. Relates to: Products (many-to-one)
9. **Orders**
   1. Stores the customer purchase details
   2. Key fields: order\_id, user\_id (FK), status, payment info
   3. Relates to: Users, Addresses, Order\_Items
10. **Order\_Items**
    1. Details of Individual products in an orders
    2. Key fields: order\_item\_id, order\_id (FK), product\_id (FK), quantity
    3. Relates to: Orders and Products
11. **Reviews**
    1. Stores the ratings/feedback
    2. Key fields: review\_id, product\_id (FK), user\_id (FK), rating (1-5)
    3. Relates to: Products and Users

**Key Relationships:**

* **Users → Orders**: One user can place many orders
* **Products → Order\_Items**: One product can appear in many orders
* **Categories → Products**: Hierarchical categorization
* **Products → Reviews**: One product can have many reviews

**Ensuring Data Integrity**

1. **Primary & Foreign Keys** – Define Primary Keys for unique identification and Foreign Keys to maintain relationships between tables, preventing orphaned records.
2. **Applying Constraints –** Enforce strict validation to maintain data consistency:
   1. **NOT NULL:** Ensures essential fields (e.g., username, email) cannot be empty.
   2. **CHECK Constraints:** Restricts values (e.g., rating between 1-5, order\_status within predefined values).
   3. **UNIQUE Constraints:** Prevents duplicate values (e.g., product\_code, email).
3. **Triggers for Automatic Updates** – Automate critical operations like inventory updates after an order is placed:

CREATE OR REPLACE TRIGGER trg\_update\_inventory

AFTER INSERT ON order\_items

FOR EACH ROW

BEGIN

UPDATE products

SET stock\_quantity = stock\_quantity - :NEW.quantity

WHERE product\_id = :NEW.product\_id;

END;

/

1. **Transactions for Consistency** – Group multiple database operations into a single transaction to maintain **ACID** properties:

Example: When an order is placed, update orders, payments, and inventory together to prevent partial updates.

**Ensuring Scalability**

1. **Horizontal Scaling (Distribute Load Across Multiple Servers)**
   * **Read Replicas** – Create read-only replicas to handle reporting queries, reducing primary database load.
   * **Sharding** – Distribute large datasets across multiple database nodes to prevent bottlenecks.
2. **Connection Pooling**
   * Use **Oracle Connection Manager** to optimize database connections and improve concurrency handling.

**Optimizing Query Performance**

1. **Using Indexes to Speed Up Queries**

Create indexes on frequently queried columns to improve search efficiency:

CREATE INDEX idx\_products\_category ON products(category\_id);

CREATE INDEX idx\_orders\_user ON orders(user\_id);

CREATE INDEX idx\_orders\_status ON orders(status);

CREATE INDEX idx\_order\_items\_order ON order\_items(order\_id);

1. **Fetch Only Required Data**

**Avoid SELECT \*** and retrieve only necessary columns for optimal performance:

SELECT product\_name, price FROM products WHERE category\_id = 10;

1. **Materialized Views for Precomputed Results**
   * Store the results of complex queries and refresh periodically to improve read performance:

CREATE MATERIALIZED VIEW mv\_top\_selling\_products AS

SELECT product\_id, SUM(quantity) AS total\_sold

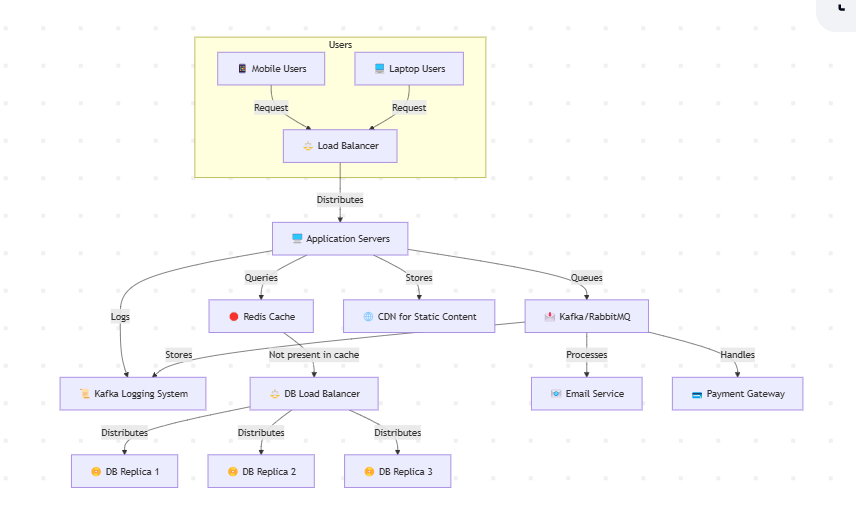
FROM order\_items

GROUP BY product\_id;

1. **Data Caching for Faster Access**

Use Oracle Result Cache or Redis to cache frequent queries like product searches, reducing database hits.

**High level ecommerce application architecture**



This architecture ensures scalability, efficiency, and fault tolerance for an e-commerce platform. Users (mobiles & laptops) send requests through a Load Balancer, which distributes them to multiple Application Servers. These servers first check Redis Cache for quick data retrieval and query the Database Replicas only if needed, improving performance. Kafka/RabbitMQ handles logging, email processing, and payment transactions asynchronously, reducing server load. Additionally, a second Load Balancer manages database replicas, and a CDN serves static content, ensuring faster load times.