DATA ENGINEERING
IT 5213
A PROJECT ON ONLINE RETAIL STORE

BY :

- JNANA SURYA PULIKANTI
- VISHWA TEJA TAMMADI LOKHANADHAM
- LEONARD LOUIS

INTRODUCTION:

- **INTRODUCTION TO DATABASES:**

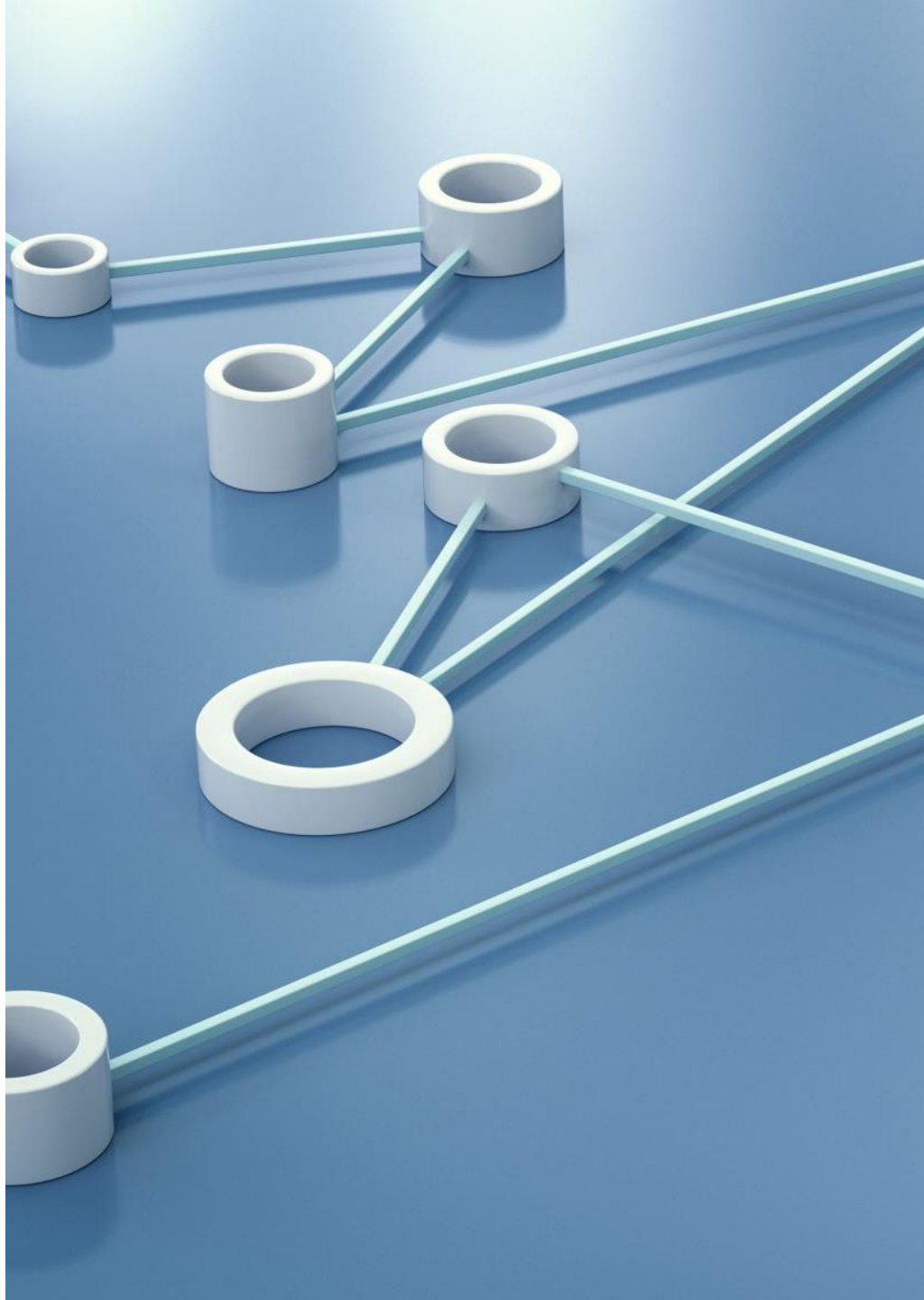
A **DATABASE** IS A STRUCTURED COLLECTION OF DATA THAT ALLOWS EFFICIENT STORAGE, RETRIEVAL, AND MANAGEMENT. IT STORES DATA IN **TABLES** CONSISTING OF ROWS AND COLUMNS, WITH **PRIMARY KEYS** TO UNIQUELY IDENTIFY EACH RECORD. EXAMPLES INCLUDE **RELATIONAL DATABASES (RDBMS)** LIKE MYSQL AND **NON-RELATIONAL DATABASES (NOSQL)** LIKE MONGODB.

- **INTRODUCTION TO NORMALIZATION:**

NORMALIZATION IS THE PROCESS OF ORGANIZING DATABASE DATA TO REDUCE REDUNDANCY AND IMPROVE INTEGRITY. IT INVOLVES BREAKING DOWN LARGE TABLES INTO SMALLER ONES AND ENSURING DATA DEPENDENCIES ARE LOGICAL. THE GOAL IS TO AVOID ANOMALIES AND MAINTAIN EFFICIENT DATA STORAGE.

- **UTILIZATION OF DATABASE FOR AN E-COMMERCE PLATFORM:**

A **DATABASE FOR AN ONLINE STORE** IS A SYSTEM USED TO STORE, MANAGE, AND RETRIEVE INFORMATION RELATED TO PRODUCTS, CUSTOMERS, ORDERS, AND TRANSACTIONS. IT ORGANIZES DATA INTO STRUCTURED TABLES, WHERE EACH TABLE REPRESENTS A SPECIFIC ENTITY, SUCH AS **PRODUCTS**, **CUSTOMERS**, **ORDERS**, AND **PAYMENTS**. FOR EXAMPLE, A **PRODUCTS** TABLE MAY STORE DETAILS LIKE PRODUCT NAME, DESCRIPTION, PRICE, AND INVENTORY QUANTITY. A **CUSTOMERS** TABLE WILL HOLD INFORMATION LIKE NAMES, ADDRESSES, AND CONTACT DETAILS. THE DATABASE ENSURES EFFICIENT MANAGEMENT OF THE STORE'S OPERATIONS, SUCH AS TRACKING INVENTORY, PROCESSING ORDERS, AND HANDLING CUSTOMER DATA, ALLOWING FOR SMOOTH ONLINE SHOPPING EXPERIENCES.



AGENDA

- E-Commerce Workflow and Unnormalized Data Challenges
- Normalization Basics for E-Commerce
- ERD Designs for core Operations
- SQL implementation with Example
- Benefits of Normalization in Business Analytics

E-COMMERCE WORKFLOW

- **KEY OPERATIONS:**

1. **USER-END:** USER REGISTRATIONS ---- CART MANAGEMENT ---- CHECKOUT ---- PAYMENT ---- ORDER FULFILLMENT
2. **WAREHOUSE-END:** INVENTORY TRACKING ---- PRODUCT UPDATES ---- RESTOCKING ALERTS

- **UNNORMALIZED DATA CHALLENGES:**

1. **DATA REDUNDANCY** : DUPLICATE PRODUCTS DETAILS IN CART AND ORDERS.
2. **MODIFICATION ANOMALIES** : ADDING, REMOVING , CHANGE IN DATA UPDATE IN ONE PLACE NOT EVERYWHERE.
3. **DATA INTEGRITY ISSUES** : ENSURING ACCURATE AND CONSISTENT DATA BECOMES DIFFICULT.
4. **COMPLEX QUERIES** : WRITING AND MAINTAINING IS HARDER DUE TO MESSY UPDATED QUERIES.
5. **POOR PERFORMANCE**: REDUNDANT DATA SLOWS DOWN UPDATES, DELETIONS, AND QUERIES.
6. **SCALING CHALLENGES**: AS DATA GROWS, MANAGING UNNORMALIZED TABLES BECOMES INEFFICIENT AND ERROR-PRONE.

- **CONCLUSION:** UNNORMALIZED DATA LEADS TO INEFFICIENCY, INCONSISTENCY, AND MAINTENANCE HEADACHES. NORMALIZATION SOLVES THESE PROBLEMS BY ORGANIZING DATA INTO CLEAN, STRUCTURED TABLES, MAKING DATABASES FASTER, SCALABLE, AND EASIER TO MANAGE.

NORMALIZATION TYPES:

- **TYPES OF NORMAL FORMS:**

1NF

- ELIMINATE TRANSITIVE DEPENDENCY

2NF

ELIMINATE PARTIAL FUNCTIONAL DEPENDENCY

3NF

ELIMINATE TRANSITIVE DEPENDENCY

BCNF

ELIMINATE FUNCTIONAL DEPENDENCY

- **USE OF 3NF NORMALIZATION IN THIS PROJECT:**

1. **PRODUCT CATALOGUE**: ENSURES A **SINGLE SOURCE OF TRUTH** FOR PRODUCT DETAILS LIKE PRICES AND CATEGORIES. THIS ELIMINATES INCONSISTENCIES AND MAKES UPDATES EASIER.
2. **USER MANAGEMENT:** PREVENTS DUPLICATE EMAILS OR ADDRESSES BY CENTRALIZING USER INFORMATION IN ONE TABLE.
3. **ORDER HISTORY:** ENABLES DYNAMIC CALCULATIONS (E.G., ORDER TOTALS) INSTEAD OF HARDCODING VALUES, ENSURING ACCURACY EVEN WHEN PRICES CHANGE.

Before 3NF

Cart Table: cart_id, user_id, product_name, price, quantity

After 3NF

Cart Table: cart_id, user_id (FK), product_id (FK), quantity

Products Table: product_id (PK), name, price, category

RELATION TO DATABASE

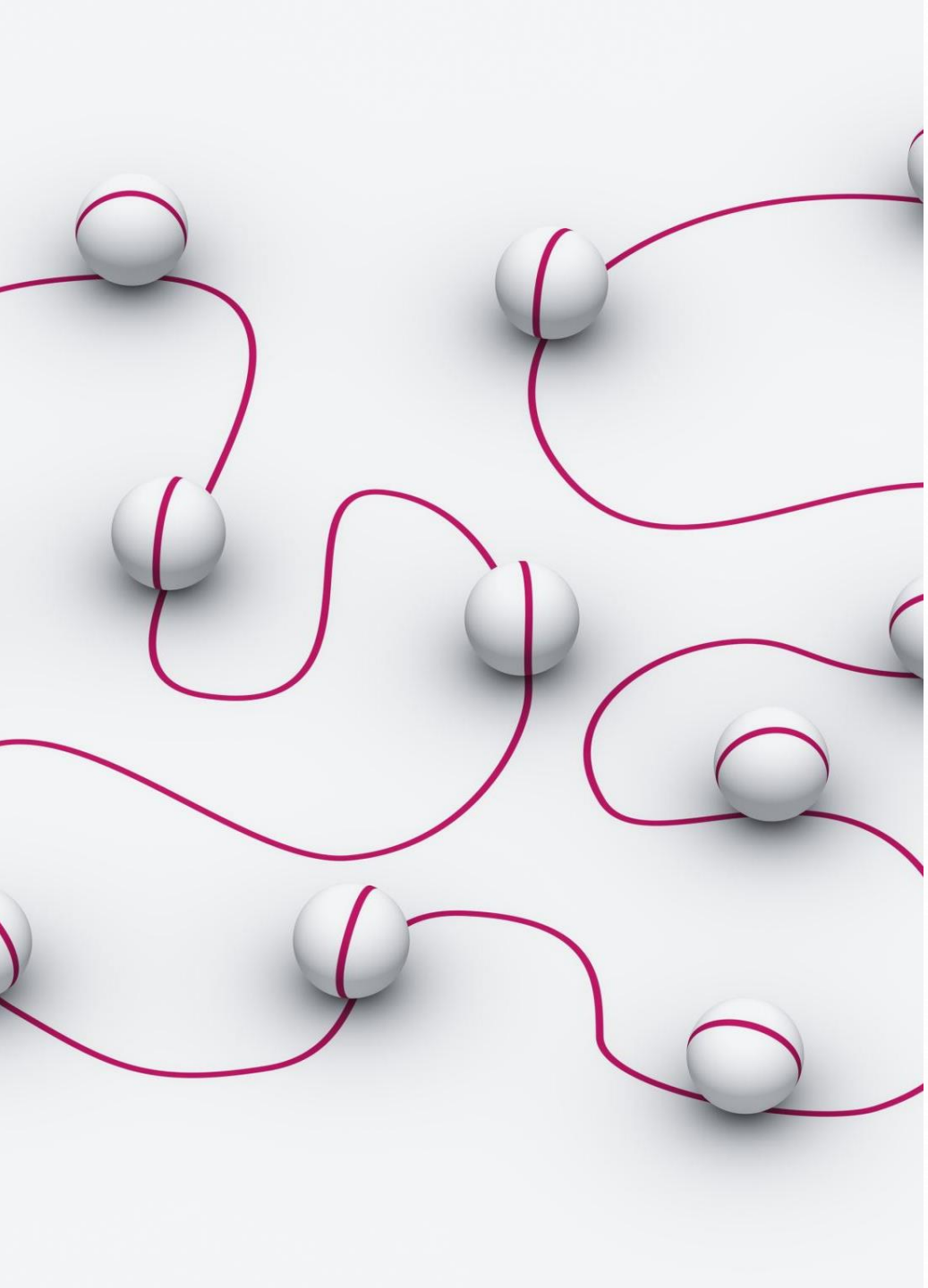
CONSISTENCY PRICING : NO MISMATCHED PRICES IN **CARTS** VS **PRODUCTS.**

REAL TIME INVENTORY: SYNC STOCK LEVELS ACROSS THE ORDERS AND RESTOCKS.

SCALABLE ANALYTICS : CLEAN DATA FOR SALES REPORTS

TABLES CREATED

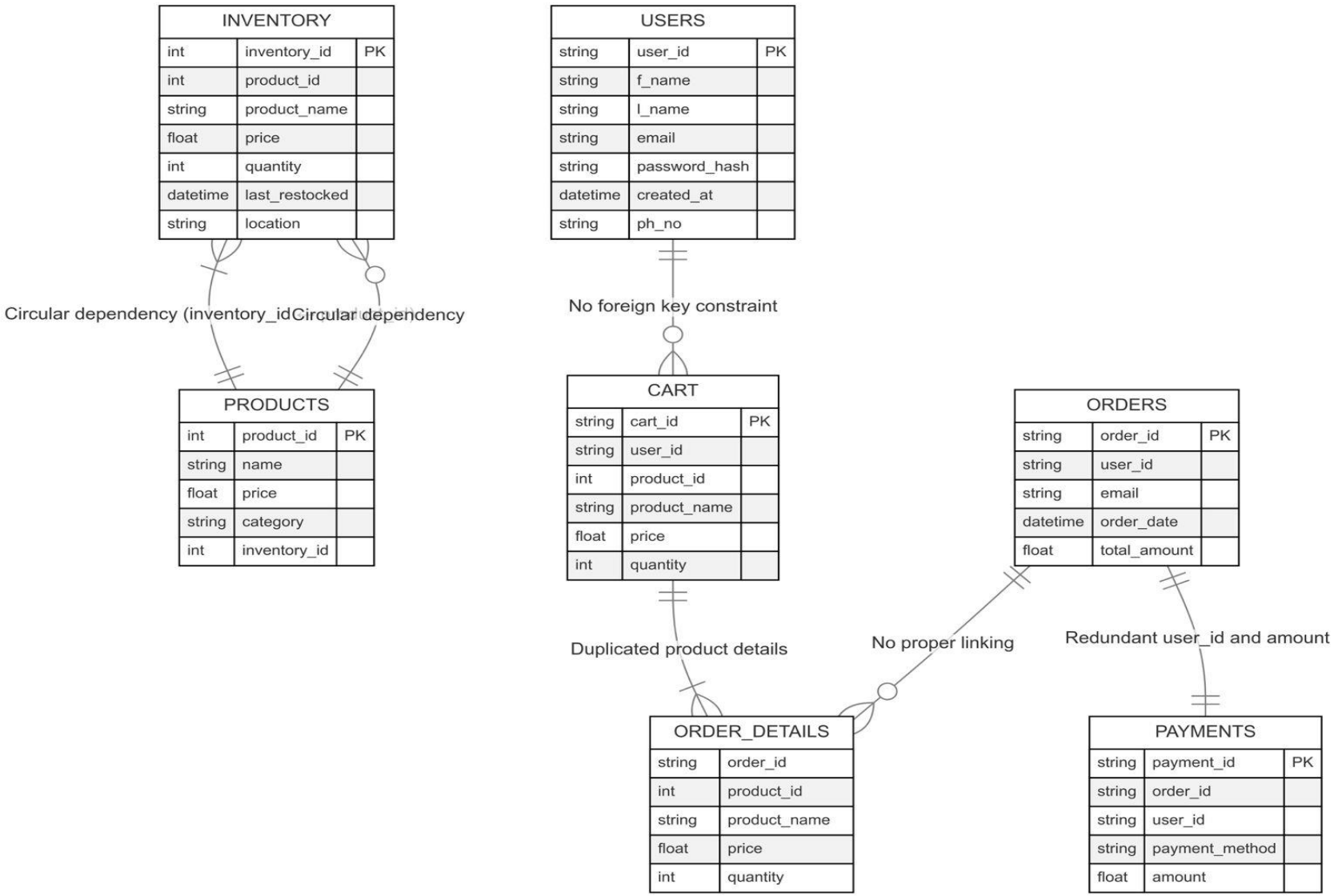
- Inventory
- Products
- Users
- User_phones
- Carts
- Cart_items
- Orders
- Order_items
- payments



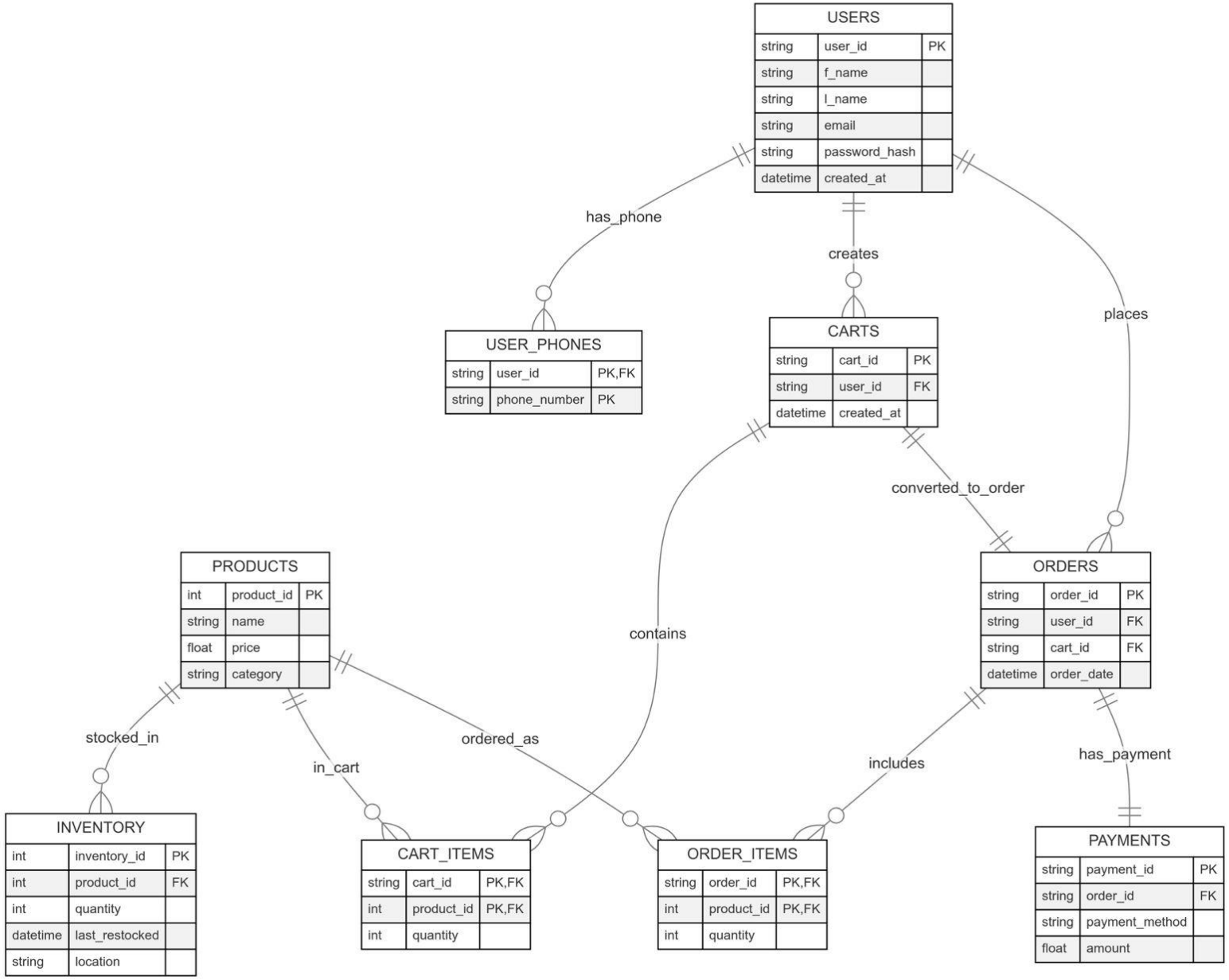
WHAT IS ERD (ENTITY RELATIONSHIP DIAGRAM) ?

- AN ERD IS A VISUAL REPRESENTATION OF THE RELATIONSHIPS BETWEEN ENTITIES IN A DATABASE.
- IT IS USED TO MODEL THE STRUCTURE OF A DATABASE AND TO SHOW THE RELATIONSHIPS BETWEEN DIFFERENT DATA ELEMENTS.
- ERD ARE ESSENTIAL FOR DESIGNING AND UNDERSTANDING THE STRUCTURE OF A DATABASE SYSTEM.
- KEY COMPONENTS OF ERD:
    1. ENTITIES
    2. ATTRIBUTES
    3. RELATIONSHIPS
    4. PRIMARY KEYS
    5. FOREIGN KEYS

ERD FOR UNNORMALIZED DATABASE



ERD AFTER NORMALIZATION



```sql
CREATE TABLE Payments (
payment_id TEXT PRIMARY KEY,
order_id TEXT NOT NULL,
payment_method TEXT NOT NULL,
amount REAL NOT NULL,
FOREIGN KEY (order_id)
REFERENCES Orders(order_id)
ON DELETE CASCADE
ON UPDATE CASCADE
);
```

```sql
CREATE TABLE Order_Items (
order_id TEXT NOT NULL,
product_id INTEGER NOT NULL,
quantity INTEGER NOT NULL,
PRIMARY KEY (order_id, product_id),
FOREIGN KEY (order_id)
REFERENCES Orders(order_id)
ON DELETE CASCADE,
FOREIGN KEY (product_id)
REFERENCES Products(product_id)
ON DELETE RESTRICT
ON UPDATE CASCADE
);
```

```sql
CREATE TABLE Products (
product_id INTEGER PRIMARY KEY,
name TEXT NOT NULL,
price REAL NOT NULL CHECK(price > 0),
category TEXT NOT NULL
CHECK(category IN ('electronics', 'home and kitchen', 'toys and games',
'sports and outdoor', 'books', 'garden and outdoors',
'baby products')) -- Fixed typo ("electricalsa" → "electronics")
);
```

```sql
CREATE TABLE Orders (
order_id TEXT PRIMARY KEY,
user_id TEXT NOT NULL,
cart_id TEXT NOT NULL,
order_date TEXT NOT NULL,
FOREIGN KEY (user_id)
REFERENCES Users(user_id)
ON DELETE RESTRICT, -- Prevent user deletion if orders exist
FOREIGN KEY (cart_id)
REFERENCES Carts(cart_id)
ON DELETE RESTRICT -- Prevent cart deletion if linked to an order
ON UPDATE CASCADE
);
```

EXAMPLES SQL QUERIES FOR CREATING TABLES IN DATABASE

ADDING IN THE DATA:

- **SYNTAX:**

INSERT INTO TABLE_NAME (COLUMN1, COLUMN2, COLUMN3, …)

VALUES (VALUE1, VALUE2, VALUE3, …);

- FOR OUT DATA BASE WE MANUALLY ENTERED DATA USING THE ABOVE CODE FOR EVERY TABLE.
- **FOR EXAMPLE:**

1. TO INSERT INTO ‘PRODUCTS’ TABLE.
2. TO INSERT INTO ‘PAYMENTS’ TABLE.

```sql
INSERT INTO Products (product_id, name, price, category) VALUES
(99245, 'logitech mouse', 34.99, 'electronics'),
(99345, 'apple pencil', 89.99, 'electronics'),
(99875, 'phillips led bulb', 22.99, 'electronics'),
(99375, 'insta pot electric kettle', 30.99, 'home and kitchen'),
(99795, 'uno cards', 10.99, 'toys and games'),
(92345, 'adidas baseball cap', 30.99, 'sports and outdoor'),
(93378, 'atomic habits', 17.99, 'books'),
(92456, 'led string lights', 21.99, 'garden and outdoors'),
(94652, 'pamper baby wipes', 18.99, 'baby products');
```

```sql
INSERT INTO Payments  VALUES
('7680990078', 'order_1001', 'credit card', 43.45),
('9490734948', 'order_2002', 'paypal', 34.87),
('9440296612', 'order_3003', 'debit card', 65.64),
('9490411662', 'order_4004', 'credit card', 57.63),
('7776640610', 'order_5005', 'credit card', 90.34),
('8886387049', 'order_6006', 'paypal', 49.50),
('9701370826', 'order_7007', 'paypal', 86.34),
('7680990076', 'order_8008', 'credit card', 53.89),
('9182948957', 'order_9009', 'credit card', 99.99);
```

SQL QUERIES FOR BUSINESS ANALYTICS

Monthly Sales Report

SELECT

strftime('%Y-%m', order_date) AS month,

SUM(p.price * oi.quantity) AS revenue

FROM Orders o

JOIN Order_Items oi ON o.order_id = oi.order_id

JOIN Products p ON oi.product_id = p.product_id

GROUP BY month;

SQL editor 4

Final Proje

Query History

```
SELECT
    strftime('%Y-%m', order_date) AS month,
    SUM(p.price * oi.quantity) AS revenue
FROM Orders o
JOIN Order_Items oi ON o.order_id = oi.order_id
JOIN Products p ON oi.product_id = p.product_id
GROUP BY month;
```

Total rows loaded: 4

	month	revenue
1	2024-11	627.91
2	2024-12	148.91
3	2025-01	350.88
4	2025-02	86.94999999999999

Identifying Best Selling Product

SELECT p.name,
SUM(oi.quantity) AS total_sold

FROM Order_Items oi

JOIN Products p ON
oi.product_id = p.product_id

GROUP BY p.name

ORDER BY total_sold DESC;

SQL editor 4

Final Proje

Query History

Explain query (F8)

```
1 SELECT p.name, SUM(oi.quantity) AS total_sold
2 FROM Order_Items oi
3 JOIN Products p ON oi.product_id = p.product_id
4 GROUP BY p.name
5 ORDER BY total_sold DESC;
```

Total rows loaded: 9

	name	total_sold
1	insta pot electric kettle	7
2	apple pencil	6
3	uno cards	5
4	pamper baby wipes	5
5	phillips led bulb	3
6	atomic habits	3
7	adidas baseball cap	3
8	logitech mouse	2
9	led string lights	1

SQL QUERIES FOR DATA MANIPULATION

Updating Product Price

UPDATE Products

SET price = 39.99

WHERE product_id = 99245;

Adding New User

INSERT INTO

Users (user_id, f_name, l_name, email, password_hash, created_at)

VALUES ('new_user', 'Alice', 'Johnson', 'alice@domain.com', 'alice@123', '2024-01-01 00:00:00');

SQL QUERIES FOR CREATING VIEWS

- **WHAT IS A VIEW?**
  A VIEW IS A **VIRTUAL TABLE** BASED ON THE RESULT OF A SQL QUERY. IT DOESN'T STORE DATA BUT DYNAMICALLY RETRIEVES IT FROM UNDERLYING TABLES WHEN QUERIED.

- **WHY USE VIEWS?**

1. **SIMPLIFY COMPLEX QUERIES**: ENCAPSULATE JOINS, FILTERS, OR CALCULATIONS.
2. **ENHANCE SECURITY**: RESTRICT ACCESS TO SPECIFIC COLUMNS OR ROWS.
3. **REUSABILITY**: SAVE AND REUSE QUERIES ACROSS APPLICATIONS.
4. **LOGICAL DATA INDEPENDENCE**: PROVIDE A CONSISTENT INTERFACE EVEN IF TABLE STRUCTURES CHANGES

To Track across all locations

CREATE VIEW Product_Stock AS

SELECT p.name, SUM(i.quantity) AS total_stock

FROM Inventory i

JOIN Products p ON i.product_id = p.product_id

GROUP BY p.name;

Final Proje View name: Product_Stock

```
1 SELECT p.name, SUM(i.quantity) AS total_stock
2 FROM Inventory i
3 JOIN Products p ON i.product_id = p.product_id
4 GROUP BY p.name
```

Filter data

	name	total_stock
1	adidas baseball cap	10
2	apple pencil	40
3	atomic habits	60
4	insta pot electric kettle	50
5	led string lights	15
6	logitech mouse	50
7	pamper baby wipes	8
8	phillips led bulb	40
9	uno cards	20

User Order History

```
CREATE VIEW User_Orders AS
SELECT
u.user_id,
u.f_name || ' ' || u.l_name AS user_name,
o.order_id,
o.order_date,
COUNT(oi.product_id) AS items_ordered
FROM Users u
JOIN Orders o ON u.user_id = o.user_id
JOIN Order_Items oi ON o.order_id = oi.order_id
GROUP BY u.user_id, o.order_id;
```

Final Proje

View name: User_Orders

```
SELECT
    u.user_id,
    u.f_name || ' ' || u.l_name AS user_name,
    o.order_id,
    o.order_date,
    COUNT(oi.product_id) AS items_ordered
FROM Users u
JOIN Orders o ON u.user_id = o.user_id
JOIN Order_Items oi ON o.order_id = oi.order_id
GROUP BY u.user_id, o.order_id
```

Grid view

Filter data

Total rows loaded: 9

	user_id	user_name	order_id	order_date	items_ordered
1	adminUser_313	michelle obama	order_7007	2024-12-25 00:00:00	1
2	buddyj_143	james watson	order_4004	2025-01-03 00:00:00	1
3	dianaM_456	diana mangi	order_8008	2025-01-13 00:00:00	1
4	emilyD_2022	emily david	order_2002	2024-11-19 00:00:00	3
5	lindaW_101	linda william	order_5005	2024-12-15 00:00:00	1
6	oliviaT_123	olivia turner	order_9009	2025-02-04 00:00:00	2
7	sophiaT_404	sophia taylor	order_6006	2025-01-04 00:00:00	2
8	techie_123	will smith	order_3003	2024-12-15 00:00:00	1
9	user_99245	john carter	order_1001	2024-11-18 00:00:00	2

ADVANCED EXTENSIONS

- FUTURE PROOFING
  - ADD DISCOUNT TABLE
  - INCLUDE SHIPPING TABLE WITH ADDRESSES AND
  - ADD REVIEWS TABLE

```
CREATE TABLE Reviews (
review_id INT PRIMARY KEY,
user_id VARCHAR(50) REFERENCES Users(user_id),
product_id INT REFERENCES Products(product_id),
rating INT CHECK (rating BETWEEN 1 AND 5)
);
```

CONTRIBUTIONS BY TEAM MEMBERS:

- DATABASE DESIGN, STRUCTURE AND CREATION- BY JNANA SURYA PULIKANTI.
- QUERY WRITING- BY JNANA SURYA PULIKANTI.
- INFORMATION GATHERING & SORTING, EXCEL TABLES ,CREATING PRESENTATION- BY VISHWA
- ERD- BY SURYA ,VISHWA, LENARDO

Q&A

Discussions

THANK YOU

"A well-structured database is the backbone of a scalable, reliable e-commerce platform!"