### A Project Report on

**Quick-Commerce Delivery System** for Database Management Systems (UCS310) By

|  |  |
| --- | --- |
| **Nitish** | **102303239** |
| **Himanshu Sardana** | **102303244** |
| **Bir Mahinder Singh** | **102303247** |
| **Agamdeep Singh** | **102303261** |

## Sub-Group: 2C21 (2C2A) Submitted to

## Dr. Geeta Kasana

****

#### DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY (A DEEMED TO BE UNIVERSITY)

#### PATIALA, PUNJAB INDIA

## Jan-May 2025

**INDEX**

[Problem Statement **3**](#_TOC_250008)

[ER Diagram **4**](#_TOC_250007)

[ER To Table (Pre-normalization) **5**](#_TOC_250006)

[ER To Table (Post-normalization) **9**](#_TOC_250005)

[Procedures, Functions & Triggers **10**](#_TOC_250004)

[SQL Query Snapshots **12**](#_TOC_250003)

[PL/SQL Snapshots **13**](#_TOC_250002)

[Conclusion **24**](#_TOC_250001)

[References **24**](#_TOC_250000)

# 

# Problem Statement

### In today’s fast-paced world, the demand for efficient, timely, and localized delivery of everyday essentials has increased significantly. Traditional retail systems often fall short in meeting this demand, especially in urban areas where consumers prefer quick doorstep delivery.

### This project aims to develop a **Quick-Commerce Delivery System** that enables customers to place orders online, which are then fulfilled by nearby dark stores (local warehouses). The system efficiently handles customer management, real-time inventory tracking, rider assignment, order processing, and payment integration. It ensures that orders are placed only when sufficient stock is available and automates rider allocation based on location and availability.

### The objective is to build a streamlined, database-driven solution that supports end-to-end order fulfillment—from placing an order to delivery—enhancing customer satisfaction and operational efficiency.

# ER Diagram

# ER To Table (Pre-normalization)

### **Customers**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| customer\_id | INT (PK) | Unique ID for each customer |
| name | VARCHAR | Customer's full name |
| phone | VARCHAR | Contact number |
| street\_no | VARCHAR | Street number |
| house\_no | VARCHAR | House or flat number |
| city | VARCHAR | City of residence |
| payment status | VARCHAR | Paid or not paid |

### **Recipients**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| recipient\_id | INT (PK) | Unique ID for recipient |
| name | VARCHAR | Recipient's name |
| phone | VARCHAR | Recipient's phone number |
| street\_no | VARCHAR | Street number |
| house\_no | VARCHAR | House or flat number |
| city | VARCHAR | City for delivery |
| customer\_id | INT | FK to Customers (who placed the order) |

### **Riders**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| rider\_id | INT (PK) | Unique ID for rider |
| name | VARCHAR | Rider's full name |
| phone | VARCHAR | Rider's contact number |
| current\_location | VARCHAR | City where rider is currently based |
| status | VARCHAR | Rider's availability status |
| rating | FLOAT | Average rider rating |
| number\_of\_reviews | INT | Count of reviews received |

### **DarkStores**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| store\_id | INT (PK) | Unique ID for dark store |
| city | VARCHAR | City where store is located |
| manager\_id | INT | FK to Managers |

### **Managers**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| manager\_id | INT (PK) | Unique ID for manager |
| name | VARCHAR | Manager's full name |
| phone | VARCHAR | Manager's phone number |

### **Products**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| product\_id | INT (PK) | Unique ID for product |
| name | VARCHAR | Product name |
| price | INT | Price per unit |
| category | VARCHAR | Product category |

### **Orders**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| order\_id | INT (PK) | Unique ID for order |
| customer\_id | INT | FK to Customers |
| recipient\_id | INT | FK to Recipients |
| rider\_id | INT | FK to Riders (nullable before assign) |
| total\_amount | INT | Final amount for the order |
| store\_id | INT | FK to DarkStores |
| order\_time | TIMESTAMP | Timestamp when order was placed |
| status | VARCHAR | Current status of the order |

### **Inventory**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| product\_id | INT | FK to Products |
| store\_id | INT | FK to DarkStores |
| stock | INT | Number of items in stock |

### **Contains**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| order\_id | INT | FK to Orders |
| product\_id | INT | FK to Products |
| quantity | INT | Quantity of product in this order |

### **Payments**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| payment\_id | INT (PK) | Unique ID for payment |
| order\_id | INT | FK to Orders |
| customer\_id | INT | FK to Customers |
| total\_amount | INT | Amount paid |
| payment\_method | VARCHAR | e.g., 'UPI', 'Card', 'Cash' |
| status | VARCHAR | e.g., 'pending', 'paid', 'failed' |
| paid\_at | TIMESTAMP | Time of payment |

# ER To Table (Post-normalization)

# All tables remain same except for Customer table and Orders table

### **Customers**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| customer\_id | INT (PK) | Unique ID for each customer |
| name | VARCHAR | Customer's full name |
| phone | VARCHAR | Contact number |
| street\_no | VARCHAR | Street number |
| house\_no | VARCHAR | House or flat number |
| city | VARCHAR | City of residence |

### **Orders**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| order\_id | INT (PK) | Unique ID for order |
| customer\_id | INT | FK to Customers |
| recipient\_id | INT | FK to Recipients |
| rider\_id | INT | FK to Riders (nullable before assign) |
| store\_id | INT | FK to DarkStores |
| order\_time | TIMESTAMP | Timestamp when order was placed |
| status | VARCHAR | Current status of the order |

# Procedures, Functions & Triggers

## Procedures:

1. **insert\_customer:** Inserts a new customer into the customers table.
2. **insert\_recipient:** Inserts a new recipient into the recipients table.
3. **insert\_rider:** Inserts a new rider into the rider table.

### **insert\_manager:** Inserts a new manager (for a dark store) into the managers table

1. **insert\_dark\_store:** Inserts a new dark store into the dark stores table.
2. **insert\_product:** Inserts a new product into the products table.

### **update\_stock:** Updates the stock for a given item at a specific dark store.

### **place\_order\_for\_self:** Takes in a customer id, item id and quantity and places an order to the nearest dark store, ensuring availability in the inventory and assigning the order to the nearest rider.

### **submit\_review:** Inserts a new review for a rider and updates the rider(s) rating.

1. **update\_order\_status:** Updates a given order’s status (from say, ‘pending’ to ‘dispatched’)

### **generate\_report:** Generates a proper report on number of orders delivered in the last 30 days, along with total revenue and average order value.

1. **get\_most\_sold\_product:** Fetches the most ordered product (by order of quantity) in the last 30 days.
2. **get\_most\_sold\_product\_by\_category:** Fetches the most ordered product (by order of quantity) of a specific category in the last 30 days.

## Functions

1. **calculate\_order\_total:** Calculates the total amount for a given order id.
2. **get\_customer\_order\_history:** Fetches a specific user’s order history via a cursor.
3. **check\_product\_availability:** Checks whether a product is available in a specific dark store or not.
4. **get\_rider\_delivery\_history:** Fetches a specific rider’s delivery history via a cursor.

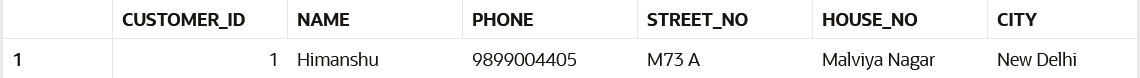
## Triggers

### **trg\_applyDiscount:** Applies a discount on the total amount of the order if it exceeds a certain threshold (eg. Rs. 1000).

# SQL Query Snapshots

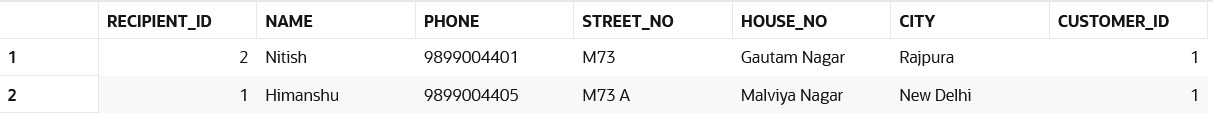
### To print all the records in the customers table

SELECT \* FROM Customers;



### To print all the records in the recipients table

SELECT \* FROM Recipients;



### To print all the records in the riders table.

SELECT \* FROM Riders;



### To print all the records in the dark stores table

##### SELECT \* FROM DARKSTORES;

##### 

# PL/SQL Snapshots

### To place an order (customer = recipient)

CREATE OR REPLACE PROCEDURE place\_order\_for\_self (

p\_customer\_id IN NUMBER, p\_item\_id IN NUMBER,

p\_quantity IN NUMBER

##### ) AS

v\_order\_id NUMBER;

v\_store\_id NUMBER;

v\_stock NUMBER;

v\_rider\_id NUMBER; v\_payment\_id NUMBER; v\_amount NUMBER;

v\_city VARCHAR2(50);

v\_price NUMBER;

##### BEGIN

SELECT city INTO v\_city FROM Customers WHERE customer\_id = p\_customer\_id;

SELECT store\_id INTO v\_store\_id FROM DarkStores

WHERE city = v\_city AND ROWNUM = 1;

SELECT rider\_id INTO v\_rider\_id FROM Riders

WHERE status = 'available' AND CURRENT\_LOCATION = v\_city AND ROWNUM = 1;

SELECT stock INTO v\_stock FROM Inventory

WHERE store\_id = v\_store\_id AND product\_id = p\_item\_id;

IF v\_stock < p\_quantity THEN

DBMS\_OUTPUT.PUT\_LINE('❌ Insufficient stock for item ID: '

|| p\_item\_id); RETURN;

##### END IF;

SELECT NVL(MAX(order\_id), 0) + 1 INTO v\_order\_id FROM Orders; INSERT INTO Orders (order\_id, customer\_id, recipient\_id,

rider\_id, store\_id, order\_time, status)

VALUES (v\_order\_id, p\_customer\_id, p\_customer\_id, v\_rider\_id, v\_store\_id, CURRENT\_TIMESTAMP, 'pending');

INSERT INTO Contains (order\_id, product\_id, quantity)VALUES (v\_order\_id, p\_item\_id, p\_quantity);

UPDATE Inventory

SET stock = stock - p\_quantity

WHERE store\_id = v\_store\_id AND product\_id = p\_item\_id;

UPDATE Riders

SET status = 'delivering' WHERE rider\_id = v\_rider\_id;

SELECT price INTO v\_price FROM Products WHERE product\_id = p\_item\_id;

v\_amount := v\_price \* p\_quantity;

SELECT NVL(MAX(payment\_id), 0) + 1 INTO v\_payment\_id FROM Payments;

INSERT INTO Payments (payment\_id, order\_id, customer\_id, amount, payment\_method, status, paid\_at)

VALUES (v\_payment\_id, v\_order\_id, p\_customer\_id, v\_amount, 'UPI', 'pending', CURRENT\_TIMESTAMP);

UPDATE Orders SET status = 'packed' WHERE order\_id = v\_order\_id;

DBMS\_OUTPUT.PUT\_LINE('✅ Order placed successfully!'); DBMS\_OUTPUT.PUT\_LINE('🛒 Order ID: ' || v\_order\_id); DBMS\_OUTPUT.PUT\_LINE('💰 Payment ID: ' || v\_payment\_id); DBMS\_OUTPUT.PUT\_LINE('🚚 Rider ID: ' || v\_rider\_id);

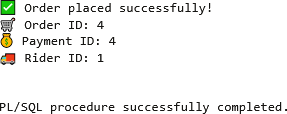
##### END;

##### BEGIN

##### PLACE\_ORDER\_FOR\_SELF(1, 1, 12);

##### END;

/



### To place and order (customer != recipient)

CREATE OR REPLACE PROCEDURE place\_order\_for\_other ( p\_customer\_id IN NUMBER,

p\_recipient\_id IN NUMBER, p\_item\_id IN NUMBER,

p\_quantity IN NUMBER

##### ) AS

v\_order\_id NUMBER;

v\_store\_id NUMBER;

v\_stock NUMBER;

v\_rider\_id NUMBER; v\_payment\_id NUMBER; v\_amount NUMBER;

v\_city VARCHAR2(50);

v\_price NUMBER;

##### BEGIN

SELECT city INTO v\_city FROM Recipients WHERE recipient\_id

= p\_recipient\_id;

SELECT store\_id INTO v\_store\_id FROM DarkStores

WHERE city = v\_city AND ROWNUM = 1;

SELECT rider\_id INTO v\_rider\_id FROM Riders

WHERE status = 'available' AND CURRENT\_LOCATION = v\_city AND ROWNUM = 1;

SELECT stock INTO v\_stock FROM Inventory

WHERE store\_id = v\_store\_id AND product\_id = p\_item\_id;

IF v\_stock < p\_quantity THEN

DBMS\_OUTPUT.PUT\_LINE('❌ Insufficient stock for item

ID: ' || p\_item\_id);

##### RETURN; END IF;

SELECT NVL(MAX(order\_id), 0) + 1 INTO v\_order\_id FROM Orders;

INSERT INTO Orders (order\_id, customer\_id, recipient\_id, rider\_id, store\_id, order\_time, status)

VALUES (v\_order\_id, p\_customer\_id, p\_recipient\_id, v\_rider\_id, v\_store\_id, CURRENT\_TIMESTAMP, 'pending');

INSERT INTO Contains (order\_id, product\_id, quantity) VALUES (v\_order\_id, p\_item\_id, p\_quantity);

UPDATE Inventory

SET stock = stock - p\_quantity

WHERE store\_id = v\_store\_id AND product\_id = p\_item\_id;

UPDATE Riders

SET status = 'delivering' WHERE rider\_id = v\_rider\_id;

SELECT price INTO v\_price FROM Products WHERE product\_id = p\_item\_id;

v\_amount := v\_price \* p\_quantity;

SELECT NVL(MAX(payment\_id), 0) + 1 INTO v\_payment\_id FROM Payments;

INSERT INTO Payments (payment\_id, order\_id, customer\_id, amount, payment\_method, status, paid\_at)

VALUES (v\_payment\_id, v\_order\_id, p\_customer\_id, v\_amount, 'UPI', 'pending', CURRENT\_TIMESTAMP);

UPDATE Orders SET status = 'packed' WHERE order\_id = v\_order\_id;

DBMS\_OUTPUT.PUT\_LINE('✅ Order placed successfully!');

DBMS\_OUTPUT.PUT\_LINE('🛒 Order ID: ' || v\_order\_id); DBMS\_OUTPUT.PUT\_LINE('💰 Payment ID: ' || v\_payment\_id); DBMS\_OUTPUT.PUT\_LINE('🚚 Rider ID: ' || v\_rider\_id);

##### END;

##### BEGIN

##### PLACE\_ORDER\_FOR\_OTHER(1, 2, 1, 5);

##### END;

##### /

##### 

### To generate a report of the most sold items in the last 30 days

CREATE OR REPLACE PROCEDURE generate\_report AS v\_total\_orders NUMBER;

v\_total\_revenue NUMBER; v\_avg\_order\_value NUMBER;

##### BEGIN

DBMS\_OUTPUT.PUT\_LINE('📊 Order Report (Last 30 Days)'); DBMS\_OUTPUT.PUT\_LINE(' ');

SELECT COUNT(\*) INTO v\_total\_orders FROM Orders

WHERE order\_time >= SYSDATE - 30;

SELECT NVL(SUM(amount), 0) INTO v\_total\_revenue FROM Payments

WHERE paid\_at >= SYSDATE - 30 AND status = 'success';

IF v\_total\_orders > 0 THEN

v\_avg\_order\_value := v\_total\_revenue / v\_total\_orders; ELSE

v\_avg\_order\_value := 0; END IF;

DBMS\_OUTPUT.PUT\_LINE('🧾 Total Orders : ' || v\_total\_orders);

DBMS\_OUTPUT.PUT\_LINE('💰 Total Revenue : ₹' ||

v\_total\_revenue);

DBMS\_OUTPUT.PUT\_LINE('📦 Avg Order Value : ₹' || ROUND(v\_avg\_order\_value, 2));

##### DBMS\_OUTPUT.PUT\_LINE(' ');

DBMS\_OUTPUT.PUT\_LINE('📌 Orders by Status:');

FOR r IN (

SELECT status, COUNT(\*) AS count FROM Orders

WHERE order\_time >= SYSDATE - 30 GROUP BY status

##### ) LOOP

DBMS\_OUTPUT.PUT\_LINE(' - ' || r.status || ': ' || r.count);

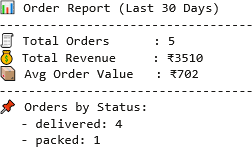
##### END LOOP; END;

##### BEGIN

##### GENERATE\_REPORT();

##### END;

/



### To get the most sold item in the last 30 days

CREATE OR REPLACE PROCEDURE get\_most\_sold\_products\_last\_30\_days AS

CURSOR most\_sold\_cursor IS SELECT c.product\_id,

p.name AS product\_name, SUM(c.quantity) AS total\_quantity\_sold

FROM Contains c

JOIN Orders o ON c.order\_id = o.order\_id

JOIN Products p ON c.product\_id = p.product\_id WHERE o.order\_time >= SYSDATE - 30

AND o.status = 'delivered' GROUP BY c.product\_id, p.name

ORDER BY total\_quantity\_sold DESC;

v\_product\_id NUMBER; v\_product\_name VARCHAR2(100); v\_total\_quantity NUMBER;

##### BEGIN

OPEN most\_sold\_cursor;

##### LOOP

FETCH most\_sold\_cursor INTO v\_product\_id, v\_product\_name, v\_total\_quantity;

EXIT WHEN most\_sold\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Product ID: ' || v\_product\_id ||

', Product Name: ' || v\_product\_name

||

', Total Quantity Sold: ' ||

v\_total\_quantity); END LOOP;

CLOSE most\_sold\_cursor;

END get\_most\_sold\_products\_last\_30\_days; BEGIN

##### GET\_MOST\_SOLD\_PRODUCTS\_LAST\_30\_DAYS();

##### END;

/



### To get the most sold grocery item in the last 30 days

##### CREATE OR REPLACE PROCEDURE

get\_most\_popular\_product\_by\_category ( p\_category IN VARCHAR2

##### ) AS

CURSOR most\_popular\_cursor IS SELECT

c.product\_id,

p.name AS product\_name, SUM(c.quantity) AS total\_quantity\_sold

FROM Contains c

JOIN Orders o ON c.order\_id = o.order\_id

JOIN Products p ON c.product\_id = p.product\_id WHERE p.category = p\_category

AND o.status = 'delivered' GROUP BY c.product\_id, p.name

ORDER BY total\_quantity\_sold DESC;

v\_product\_id NUMBER; v\_product\_name VARCHAR2(100); v\_total\_quantity NUMBER;

##### BEGIN

OPEN most\_popular\_cursor;

##### LOOP

FETCH most\_popular\_cursor INTO v\_product\_id, v\_product\_name, v\_total\_quantity;

EXIT WHEN most\_popular\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Product ID: ' || v\_product\_id ||

', Product Name: ' || v\_product\_name

||

', Total Quantity Sold: ' ||

v\_total\_quantity); END LOOP;

CLOSE most\_popular\_cursor;

END get\_most\_popular\_product\_by\_category;

##### BEGIN

GET\_MOST\_POPULAR\_PRODUCT\_BY\_CATEGORY('grocery');

##### END;

/



### To print the order history for a particular customer

##### CREATE OR REPLACE FUNCTION

get\_customer\_order\_history(p\_customer\_id IN NUMBER) RETURN SYS\_REFCURSOR

##### AS

v\_cursor SYS\_REFCURSOR;

##### BEGIN

OPEN v\_cursor FOR

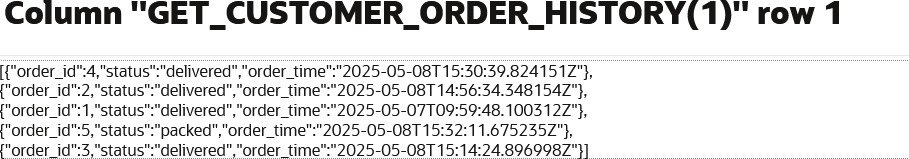
SELECT o.order\_id, o.status, o.order\_time FROM Orders o

WHERE o.customer\_id = p\_customer\_id;

RETURN v\_cursor;

END get\_customer\_order\_history;

SELECT GET\_CUSTOMER\_ORDER\_HISTORY(1) FROM dual;



### To print the delivery history for a particular rider

##### CREATE OR REPLACE FUNCTION

get\_rider\_delivery\_history(p\_rider\_id IN NUMBER) RETURN SYS\_REFCURSOR

##### AS

v\_cursor SYS\_REFCURSOR; BEGIN

OPEN v\_cursor FOR

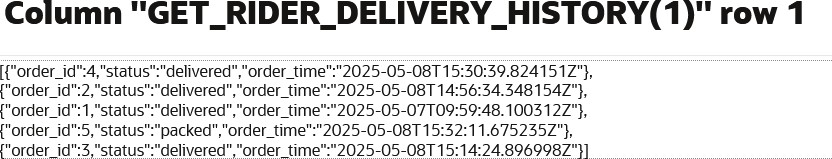
SELECT o.order\_id, o.status, o.order\_time FROM Orders o

WHERE o.rider\_id = p\_rider\_id;

RETURN v\_cursor;

END get\_rider\_delivery\_history;

SELECT GET\_RIDER\_DELIVERY\_HISTORY(1) FROM dual;



# Conclusion

### This project delivers a comprehensive and scalable **Quick Commerce Delivery System** designed to simulate the backend operations of a modern hyperlocal delivery platform. In today’s fast-paced urban environment, the demand for near-instant delivery of groceries, daily essentials, and other products has given rise to the quick commerce model. This system captures the essential components required to operate such a platform effectively, including customer management, inventory tracking, order processing, and payment handling.

### The database design is grounded in real-world use cases and follows normalization principles to ensure data integrity and efficiency. Key entities such as **Customers**, **Recipients**, **Riders**, **Dark Stores**, **Products**, **Orders**, and **Payments** are clearly defined and interconnected via appropriate constraints. This structured schema ensures smooth operation, traceability of orders, and reliable reporting.

### One of the strengths of the system lies in its modularity and extensibility. It seamlessly manages complex relationships between various components like customer profiles, delivery riders, order contents, inventory levels, and payment transactions. Integrity constraints and relational consistency are strictly enforced to ensure that each record in the database is valid and meaningful, reducing the likelihood of errors or data anomalies.

### In addition, the system supports business logic such as **dynamic discount application** for high-value orders through the use of database triggers. This feature demonstrates the system's capability to incorporate promotional strategies and real-time business decisions at the data layer

### itself, allowing for flexible and responsive behavior based on order context.

## Future Enhancements

### **Order History & Analytics:** Build a dashboard for customers and managers to view order history, top products, repeat purchases, and delivery performance.

**Coupon & Loyalty System**: Integrate discount coupons and customer loyalty points to boost engagement and retention.

### **Delivery Time Estimation**: Implement algorithms that calculate expected delivery times based on distance, rider availability, and store processing speed.

# 

# References

* <https://www.geeksforgeeks.org/cursors-in-pl-sql/>
* <https://www.geeksforgeeks.org/plsql-triggers/>
* <https://www.w3schools.com/sql/sql_join.asp>
* <https://medium.com/@singourhimanshu3/high-level-system-design-for-blinkit-instamart-a-scalab> [le-and-efficient-model-edf4300b09ed](https://medium.com/%40singourhimanshu3/high-level-system-design-for-blinkit-instamart-a-scalable-and-efficient-model-edf4300b09ed)
* [https://www.geeksforgeeks.org/how-to-design-a-database-for-10-minute-grocery-delivery-app-lik](https://www.geeksforgeeks.org/how-to-design-a-database-for-10-minute-grocery-delivery-app-like-zepto/) [e-zepto/](https://www.geeksforgeeks.org/how-to-design-a-database-for-10-minute-grocery-delivery-app-like-zepto/)