blinkit

DBMS PROJECT

Work In progress

Prepared By : Group-7

Abhishek - 2022020

Krishnendu - 2022255

Rahul - 2022389

What is blinkit ?

Single App to fulll fill your all daily needs.

we are the onestop solution to get groceries and other items in home !

we provide online groceries , medicine , stationaries , food items delivery for 24×7 we delivered fastest and safetly in 10 mins .

Aim

We are creating a database for b2c ,online retail stores where we will keep data of grocery items present in our warehouse , People will be able to buy items and it will be delivered to customers in a few minutes.

Where customer can place order in our website or app we can deliver in few minutes and customer will also have the options to cancel the order or change the order as per thier needs . If it’s refundable item then we credit bank to customer id .

Customer have to provide us Basic detail to regsiter like phoneNumber , Address and payment details , we can store address partwise in database to make use out off it.

we are partnering with groceries store to get item from instant when customer needs and then we can deliver and also store some important groceries , fruits & Vegetables to provide instantly.

we are also partnering from delivery boys on per delivery charge basis there we will provide our bags and team kit to them .

Where If Vendors can regsiter there with basic details of registration and details like adddress , phoneNumber , gstNumb , etc after thier product can be list , where if customer order from nearest range to the seller then our delivery partner can collect and we will pay them as per order wise .

Customer can also use benefits of firstTime order 50% off and some festivals sale , coupons to get benefits and perks.

Also , User can contact customer support for getting defected and expiry product wherther we will provide 100% or partwise compensation as per product wise .

STAKEHOLDERS

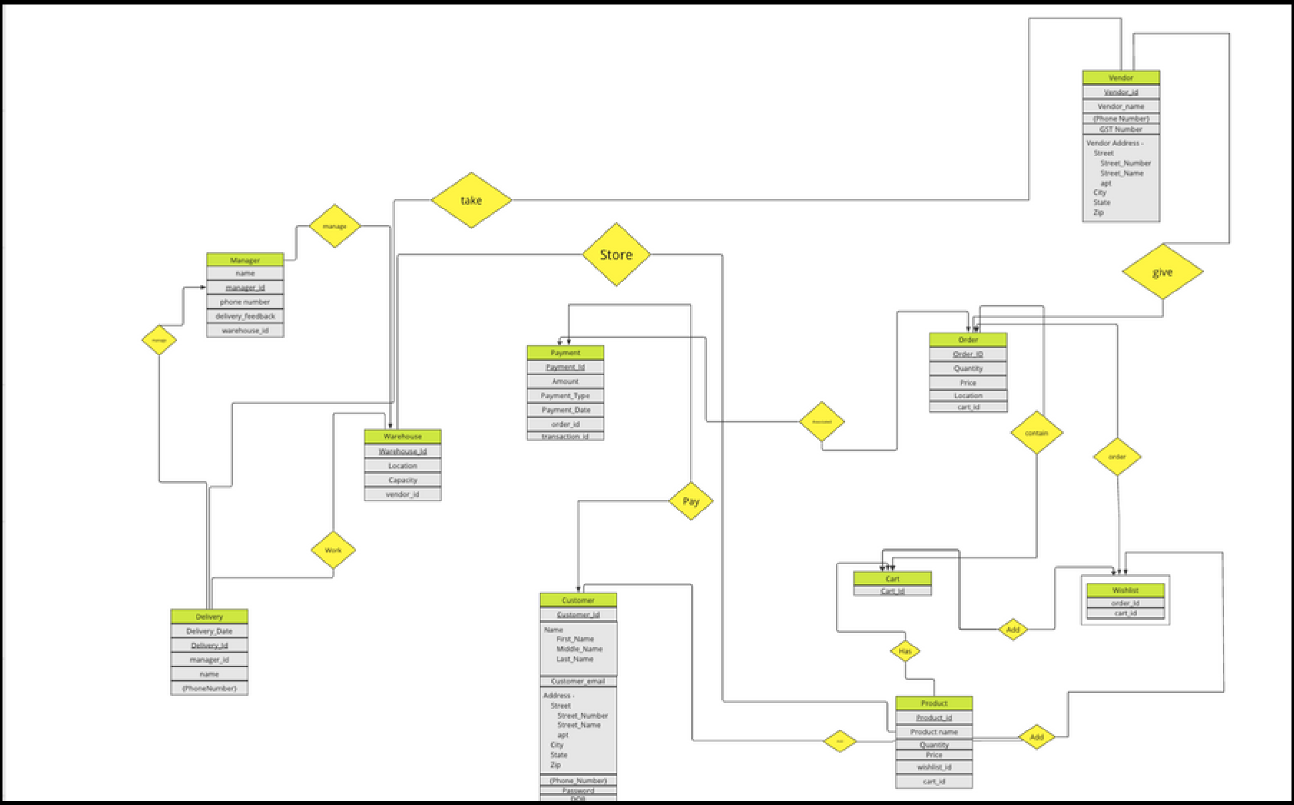
- Users- customers who uses for daily needs and give valuable feedbacks.
- Recuriter - Who appoints warehouse , delivery boys , shops , etc.
- Developer - Who manage database , servers App and mobile Apps Version and so on .
- Customer Support - Team for providing assitance to customer related queries and assurance .

Contribution -Group 7

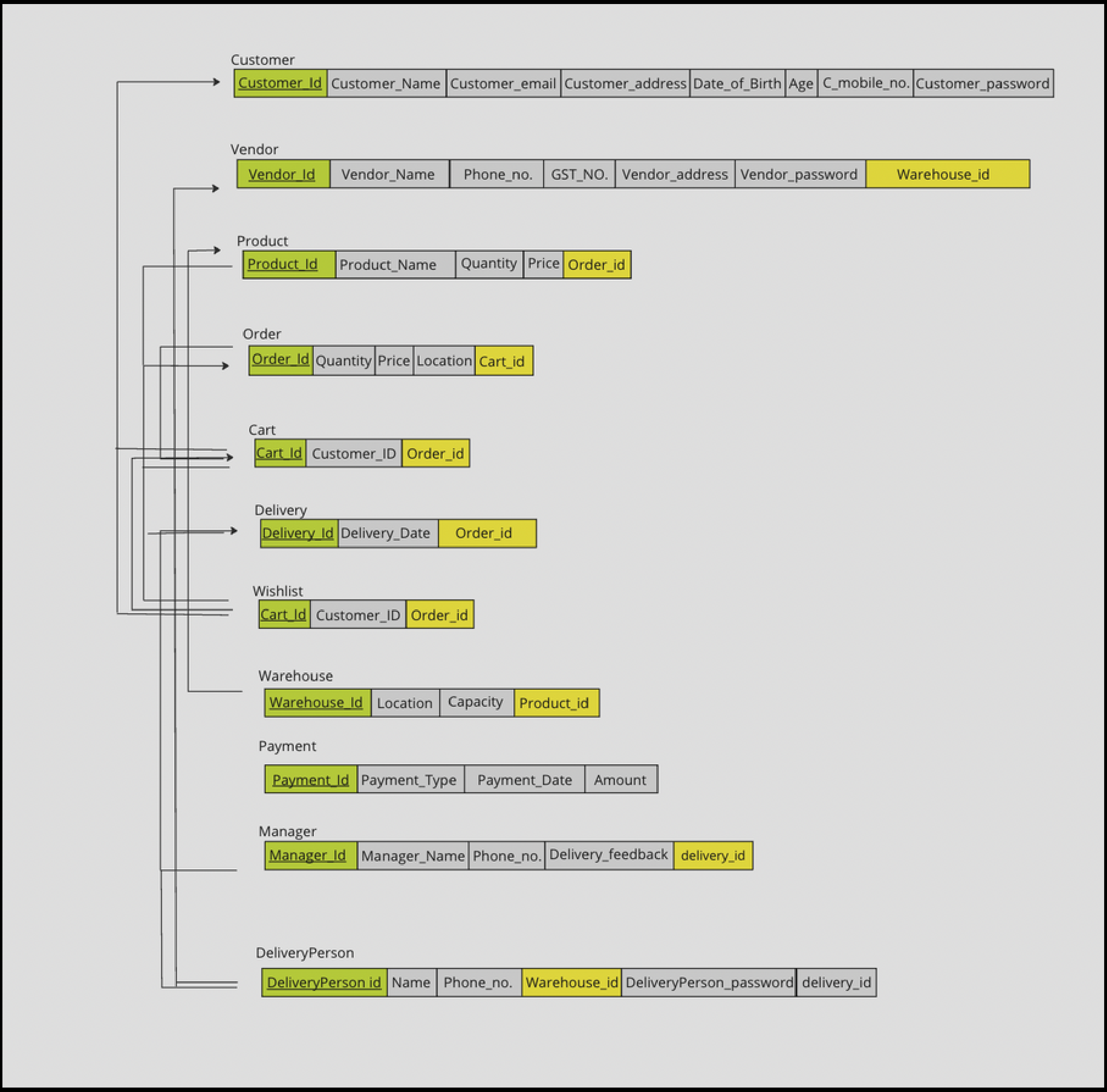
Abhishek - Researching Bussiness Model
Rahul - Convert Research in Slides + Stakeholders
Krish - Comparision and Benefits + Slides.
All Contributed Equally + **NO AI CHATGPT used**

Week-2

ER Diagram



Relational Model


Customer
Customer_Id | Customer_Name | Customer_email | Customer_address | Date_of_Birth | Age | C_mobile_no. | Customer_password
Vendor
Vendor_Id | Vendor_Name | Phone_no. | GST_NO. | Vendor_address | Vendor_password | Warehouse_id
Product
Product_Id | Product_Name | Quantity | Price | Order_id
Order
Order_Id | Quantity | Price | Location | Cart_id
Cart
Cart_Id | Customer_ID | Order_id
Delivery
Delivery_Id | Delivery_Date | Order_id
Wishlist
Cart_Id | Customer_ID | Order_id
Warehouse
Warehouse_Id | Location | Capacity | Product_id
Payment
Payment_Id | Payment_Type | Payment_Date | Amount
Manager
Manager_Id | Manager_Name | Phone_no. | Delivery_feedback | delivery_id
DeliveryPerson
DeliveryPerson id | Name | Phone_no. | Warehouse_id | DeliveryPerson_password | delivery_id

Contribution

Abhishekh - ER Diagram

KrishNendu - relational model

Rahul- corrections in er & relational model

miro- https://miro.com/app/board/uXjVN09FxGI=/

Week -3

Customer

Customer_Id	Primary key
Name	Varchar
Email	Varchar
Dob	Int
PhoneNumber	Varchar
Password	Varchar
Age	Int

Wishlist

Order_Id	Foreign key
Cart_Id	Foreign key

Cart

Cart_id	Primary key

Order

Order_Id	Primary key
total_amout	Int
Quantity	Int
Location	Int
Cart_Id	Foreign key

Product

Product_Id	Primary key
Price	Int
Quantity	Int
Product_name	Int
Order_Id	Foreign key

Warehouse

Warehouse_id	Primary key
Vendor_id	Foreign key
Capacity	Int
Location	Varchar

Delivery

Delivery_id	Primary key
Manager_id	Foreign key
Vendor_id	Foreign key
name	Varchar
Phone number	Int

Manager

manager_id	Primary key
warehouse_id	Foreign key
name	Int
phone_number	Int
feedback	Varchar

Vendor

vendor_id	Primary key
GST	Varchar
name	Int
phone_number	Int
addresss	Varchar

Sql Queries

1.) Find products with greater than the average quantities across all orders

```
SELECT p.product_id, p.product_name, p.quantity
FROM Product p
WHERE p.quantity >
(SELECT AVG(quantity)
FROM Orders);
```

2.) Find the top 3 cities with the most customers and their average age:

```
SELECT city, COUNT(User_ID) as
 Number_of_Customers,AVG(age) as Average_Age
FROM Customer
GROUP BY cityORDER BY Number_of_Customers DESC
LIMIT 3;
```

3.) Customer who have not added item to wishlists::

```
SELECT User_ID, first_name, last_name
FROM Customer
WHERE User_ID NOT IN (
   SELECT DISTINCT cart_id
   FROM Wishlist);
```

4.) Find the first name , last name who works as customer or either vendor:

```
SELECT first_name, last_name, 'Customer' AS role
FROM Customer)
UNION ALL
(SELECT first_name, last_name, 'Vendor' AS role
FROM Vendor);
```

5.) Find the customers who live in the city with the most customers:

```
SELECT *
FROM Customer
WHERE city = (
   SELECT city
   FROM (
      SELECT city, COUNT(User_ID) as Number_of_Customers
      FROM Customer
      GROUP BY city
      ORDER BY Number_of_Customers DESC
      LIMIT 1
   ) AS MostPopulatedCity
);
```

6.) Find the customers who live in the city with the highest average age:

```
SELECT *
FROM Customer
WHERE city = (
   SELECT city
   FROM (
      SELECT city, AVG(age) as Average_Age
      FROM Customer
      GROUP BY city
      ORDER BY Average_Age DESC
      LIMIT 1
   ) AS CityWithHighestAvgAge
);
```

7.) Find the product with greater than the average quantities across all orders:

```
SELECT p.product_id, p.product_name, p.quantity
FROM Product p
WHERE p.quantity > (
   SELECT AVG(quantity)
   FROM Orders
);
```

8.) Find the customers who have placed orders in more than one location:

```
SELECT c.User_ID, c.first_name, c.last_name
FROM Customer c
WHERE c.User_ID IN (
   SELECT User_ID
   FROM (
      SELECT User_ID, COUNT(DISTINCT location) AS num_locations
      FROM Orders
      GROUP BY User_ID
   ) AS subquery
   WHERE num_locations > 1
);
```

9.) Find the number of order for each customer:

```
SELECT Customer.first_name, Customer.last_name,
COUNT(Orders.order_id) AS num_orders
FROM Customer
JOIN Orders ON Customer.User_ID = Orders.cart_id
GROUP BY Customer.User_ID;
```

10.) find the customer who live in same city as a vendor:

```
SELECT c.User_ID, c.first_name, c.last_name, c.city
FROM Customer c
JOIN Vendor v ON c.city = v.city;
```

Customer Features

1.) We have implemented a **login / signup** page for the customers.

2. While s**igning up,** we are presented with all the details that the customer needs to fill.

3. For **logging in**, we take inputs of the customer_id, customer_username and the customer's password. The customer_id shall be used everytime we try to achieve a particular task related to the customer.

4. If the customer's username and password matches with the data in our database, we will enter the customer menu.

5. The customer menu has about 21 options where one option is to exit the menu.

6. **Change Personal Details:** It change personal details whether it's the username, password, date of birth, primary phone number, secondary phone number, email address, city or state.

7. **View Categories:** It allows us to view all the distinct categories available in Blinkit database.

8. **View Products:** It allows us to view all the products available in Blinkit database.

9. **View Cart** : It allows us to view the cart of the particular customer who has logged in.

10. **Add Products to Cart**: It allows us to add a specific product into the customer's cart based on the product_id which the customer has to provide.

11. **Add Money to Wallet**: It allows us to add money to the wallet of the customer.

12. **Check wallet Balance**: It allows the customer to view the remaining balance in his/her wallet

13. **Remove Product from Cart**: It allows the customer to remove a particular product from the cart.

14. **View Orders:** It allows the customer to view all the orders he/she has in his/her orders list.

15. **Checkout**: It allows the customer to checkout all the products from the cart. It also handles cases where the customer must have the required amount of money in his/her wallet for the checkout to happen and the amount of products are also consequently reduced in the inventory.

16. **View Delivery Status**: It allows the customer to see how many days it will take for the order to be delivered.

17. **Empty Cart:** It allows the customer to empty his/her cart.

18. **View Product Reviews**: It allows the customer to view all the product reviews.

19. **View your product reviews**: It allows the customer to view the product reviews which he/she has made.

20. **View Wishlist:** It allows the customer to view the wishlist that he/she has created.

21. **Add Product to Wishlist**: It allows the customer to add products inside his/her wishlist.

22. **Clear Wishlist**: It allows the customer to clear his/her wishlist.

Seller Features

On continuing to enter as a seller, the following options :

1. Login / Signup : we have implemented a login/signup page for the seller

2. To login, the seller need to fill in their seller_id, seller_username, seller_password.

3. To Signup, the seller has to input all his details to register his account on the website.

4. If he chooses to login he is further presented with the seller features, if he chooses to signup, after creating his account he is required to login and is presented with the features.

5. The features include : ● Search category : the user can search all the categories available on the website.

● Search product : the user can search all the products available on the website.

● Add Category: as a seller, a user can add any category to the database which will be available to all other users on the website.

● Add product : as a seller, a user can add any product under any existing category which will be available for all the users to view.

● Assign delivery agent: the user can assign a delivery agent to any existing order of his products.

● Show all running orders : the user can look at his running orders which have not ben delivered yet. ●

Triggers

1. **Block User Login Trigger**: If some user enter 4 times wrong pass then user get block for 30 seconds .

```
CREATE TRIGGER block_user_login
AFTER INSERT ON django_admin_log
FOR EACH ROW
BEGIN
    DECLARE recent_attempts INT;
    DECLARE last_attempt TIMESTAMP;
    DECLARE blocked_until TIMESTAMP;

    -- Get the count of recent failed login attempts for the user
    SELECT COUNT(*), MAX(action_time)
    INTO recent_attempts, last_attempt
    FROM django_admin_log
    WHERE action_flag = 1  -- Failed login attempt
    AND username = NEW.username
    AND action_time > NOW() - INTERVAL 1 MINUTE;  -- Within the last
minute

    -- If three consecutive failed attempts within 1 minute
    IF recent_attempts >= 3 THEN
        -- Set the blocked_until timestamp to 30 seconds from the last
attempt
        SET blocked_until = last_attempt + INTERVAL 30 SECOND;

        -- Update the user's record to indicate blocked status
        UPDATE auth_user
        SET is_blocked = 1, blocked_until = blocked_until
        WHERE username = NEW.username;
    END IF;
END;
```

2.) **Update Price Trigerr:** add product in wishlist then it add product instanly then update the product price and quantitty.

```
CREATE TRIGGER update_cart_price_after_insert
AFTER INSERT ON ecommerceapp_cart_item
FOR EACH ROW
BEGIN
    DECLARE total_price DECIMAL(10, 2);

    -- Calculate the total price of all items in the cart
    SELECT SUM(p.price * ci.quantity)
    INTO total_price
    FROM ecommerceapp_cart_item ci
    JOIN ecommerceapp_product p ON ci.product_id = p.id
    WHERE ci.cart_id = NEW.cart_id;

    -- Update the total price in the cart table
    UPDATE ecommerceapp_cart
    SET total_price = total_price
    WHERE id = NEW.cart_id;
END;
```

3.) **After failed login:** if user give wrong email which is not registered previously then it blocks after 4 attempts.

```
CREATE TRIGGER after_failed_login
AFTER INSERT ON login_attempts
FOR EACH ROW
BEGIN
   DECLARE email_count INT;

   -- Check if the email exists in the database
   SELECT COUNT(*)
   INTO email_count
   FROM users
   WHERE email = NEW.email;

   -- If the email doesn't exist, call the stored procedure to block it
   IF email_count = 0 THEN
      CALL block_email(NEW.email);
   END IF;
END;
```

Non Conflict Transactions

```
-- T1 transaction

START TRANSACTION;
INSERT INTO Customer (first_name, last_name, email, street_no,
street_name, apt, city, state, zip_code, phone_no, password, age,
Gender)
VALUES ('Mark', 'Johnson', 'mark.j@example.com', '2022', 'Cedar
Street', 'Apt 505', 'Villageton', 'FL', '90123', '555-6709',
'password8', 28, 'Male');
COMMIT;
```

```
-- T2 transaction

Start Transactions;
UPDATE Manager
SET delivery_feedback = 'Excellent'
WHERE manager_id = 3;
Commit;
```

```
-- T3 transaction
START TRANSACTION;
SELECT manager_id, name, phone_no, delivery_feedback
FROM Manager
WHERE warehouse_id = 5;
COMMIT;
```

```
-- T4 transaction

Start Transactions;
INSERT INTO Customer (first_name, last_name, email, street_no,
street_name, apt, city, state, zip_code, phone_no, password, age,
Gender)
VALUES ('Laura', 'Williams', 'lauraa.w@example.com', '3030', 'Elm
Avenue', 'Apt 606', 'Metropolis', 'IL', '34567', '555-3796',
'password9', 35, 'Female');
Commit;
```

Explanations

As it serializable schedule there , we can see each transaction
occurs independently there we can see no conflict there
the serialiable order of t4 -> t1 -> t2 -> t3

Timestamps

Time	T1	T2	T3	T4
1				begin
2				read(customer)
3				write(customer)
4				commit
5	begin			
6	read(customer)			
7	write(customer)			
8	commit	begin		
9		read(Manager)		
10		Write(Manager)		
11		commit	begin	
12			read(Manager)	
13			commit	

```sql
-- T1 transaction

START TRANSACTION;
INSERT INTO Customer (first_name, middle_name, last_name,
email, street_no, street_name, apt, city, state, zip_code, phone_no,
password, age, Gender)
VALUES ('Mark', 'S', 'Johnson', 'mark.j@example.com', '2425',
'Chestnut Street', 'Apt 707', 'Riverdale', 'IL', '45678', '555-4321',
'password11', 30, 'Male');
COMMIT;


-- T2  transaction

START TRANSACTION;
UPDATE Vendor SET Opening_Time = '07:30:00' WHERE vendor_id = 6;
COMMIT;



-- T3 transaction

START TRANSACTION;
INSERT INTO Payment (payment_date, payment_type, amount,
transaction_id)
VALUES ('2024-04-20', 'Credit Card', 55.00, 109);
COMMIT;


-- T4 transaction

START TRANSACTION;
INSERT INTO Product (product_name, quantity, price, wishlist_id)
VALUES ('Stylish Sunglasses', 5, 29.99, 2);
COMMIT;
```

Timestamps

Time	T1	T2	T3	T4
1	begin			
2	read(customer)			
3	write(customer)			
4	commit			
5		begin		
6		read(Vendor)		
7		**write(Vendor)**		
8		commit		
9			begin	
10			read(Payment)	
11			write(Payment)	
12			commit	
13				begin
14				read(Product)
15				write(Product)
16				commit

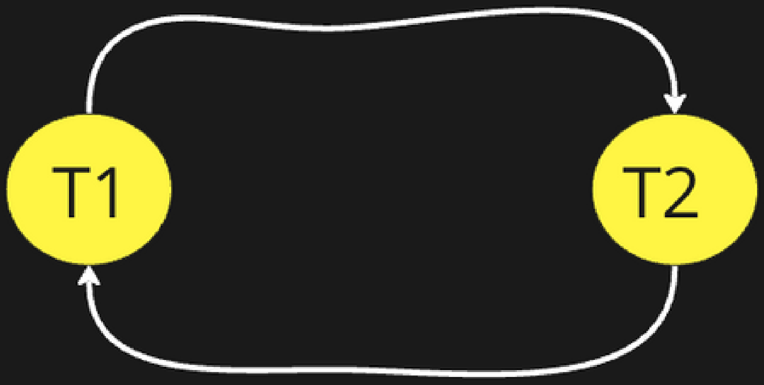
Conflicting

```
-- T1  transaction

 START TRANSACTION;
UPDATE Product SET quantity = quantity + 10 WHERE product_id = 1;
UPDATE Product SET quantity = quantity - 5 WHERE product_id = 1;
COMMIT;


-- T2  transaction

 START TRANSACTION;
UPDATE Product SET quantity = quantity + 20 WHERE product_id = 1;
UPDATE Product SET quantity = quantity - 15 WHERE product_id = 1;
COMMIT;
```



Timestamps

Time	T1	T2	Value
1	begin		0
2	read(product_quantity)		0
3	quantity = quantity + 10		0
4	write(quantity)	begin	10
5		read(product_quantity)	10
6		quantity = quantity + 20	10
7		write(quantity)	20
8	read(product_quantity)		20
9	commit		20
10		commit	20

Explanations

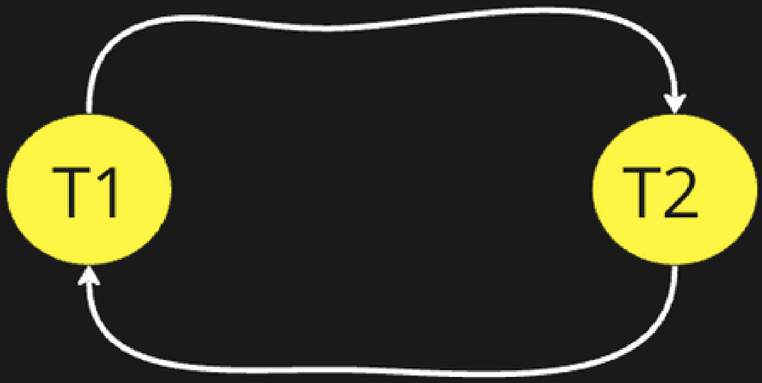
WR - write read conflicts where we read quantity from t1 then write in t2 which take updated values from t1 .
RW - read write conflicts where make reads from t2 from updated values of quantity there and the read in t1 make them commited.

```
-- T1  transaction

 START TRANSACTION;
UPDATE Orders SET quantity = 2 WHERE order_id = 1;
UPDATE Orders SET quantity = 8 WHERE order_id = 1;
COMMIT;
```

```
-- T2  transaction

 START TRANSACTION;
UPDATE Orders SET quantity = 5 WHERE order_id = 1;
COMMIT;
```



Timestamps

Time	T1	T2	Value
1	begin		0
2	read(order_quantity)		0
3	quantity = 2		0
4	write(quantity)	begin	2
5		read(order_quantity)	2
6		quantity = 5;	2
7		write(quantity)	5
8	read(order_quantity)		5
9	quantity = 8		5
10	write(quantity)		8
11	commit		8
12		commit	8

Explanations

WR - write read conflicts where we read quantity from t1 then write in t2 which take updated values from t1 .
RW - read write conflicts where make reads from t2 from updated values of quantity there and the read in t1 make them commited.