

Project 2

Ahmed Tlili, Leon Petrinis, Mathilde Peruzzo

March 7, 2025

1 ER Diagram

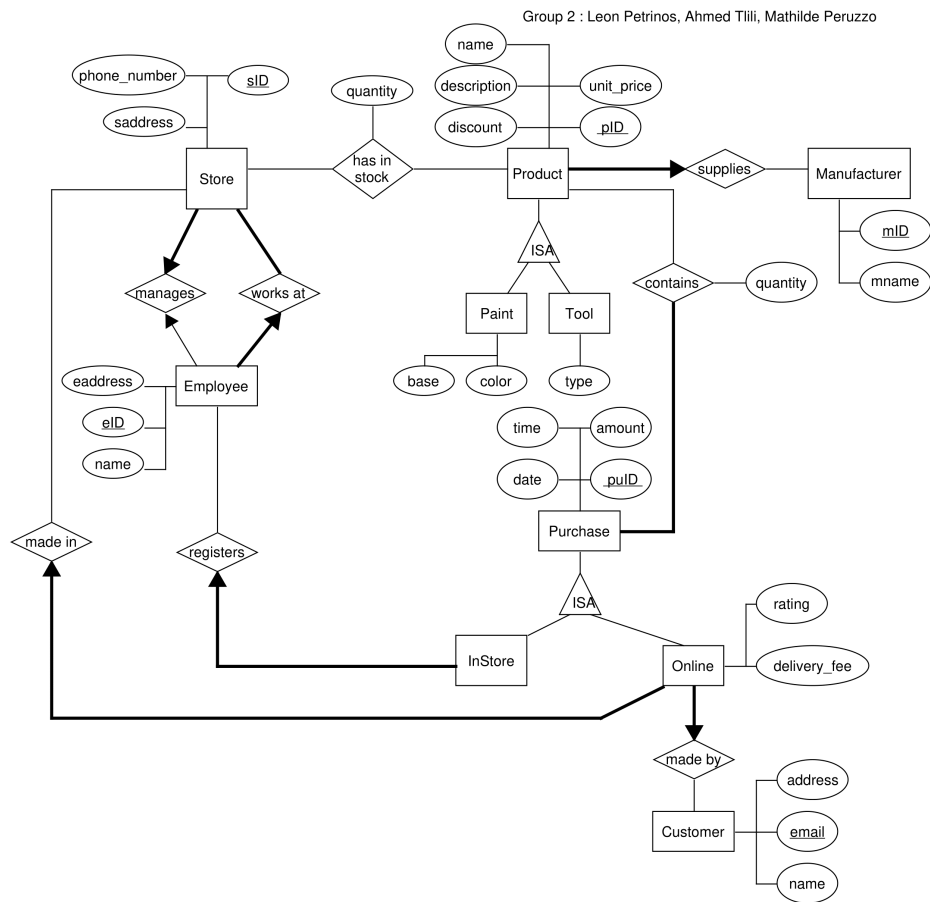


Figure 1: ER Diagram

2 Relational Schema

- **Store**(s_id, s_address, phone_number, manager_id UNIQUE NOT NULL)
FOREIGN KEY(manager_id) REFERENCES Employee(employee_id)
- **Employee**(e_id, e_name, s_id)
FOREIGN KEY(s_id) REFERENCES Store(s_id)
- **Manufacturer**(m_id, m_name)

- **Product**(p_id, p_name NOT NULL, unit_price NOT NULL, description, discount_percentage, m_id NOT NULL)
FOREIGN KEY(m_id) REFERENCES Manufacturer(m_id)
- **Paint**(p_id, base, color)
FOREIGN KEY(p_id) REFERENCES Product(p_id)
- **Tool**(p_id, type)
- **Has_in_stock**(p_id, s_id, quantity NOT NULL CHECK(quantity ≥ 0))
FOREIGN KEY(p_id) REFERENCES Product(p_id)
FOREIGN KEY(s_id) REFERENCES Store(s_id)
- **Customer**(email, c_name, c_address NOT NULL)
PRIMARY KEY(email)
- **Purchase**(p_id, amount NOT NULL, p_date NOT NULL, p_time NOT NULL)
- **Contains_purchase**(p_id, product_id, quantity NOT NULL CHECK(quantity ≥ 0))
FOREIGN KEY(p_id) REFERENCES Purchase(p_id)
FOREIGN KEY(product_id) REFERENCES Product(p_id)
- **Instore**(p_id, e_id)
FOREIGN KEY(p_id) REFERENCES Purchase(p_id)
FOREIGN KEY(e_id) REFERENCES Employee(e_id)
- **Online**(p_id, rating CHECK(rating ≥ 0 AND rating ≤ 5 OR rating IS NULL), delivery_fee NOT NULL, email NOT NULL)
FOREIGN KEY(p_id) REFERENCES Purchase(p_id)
FOREIGN KEY(email) REFERENCES Customer(email)

3 Pending Constraints

- A store should have at least one employee. (TODO: might not be correct as a every store should have a manage which will work there as well)
- A purchase should have at least one product.
- Cannot have store manager_id referencing a row in the Employee table.
As here we have two tables referencing each other (STORE, EMPLOYEE).
One of them has to drop the foreign key constraint.

4 SQL Queries

Query 1

- List the id and address of every store with the respective quantities of the products (with p_id = 3) they have in stock.
-

```
SELECT STORE.s_id , s_address , COALESCE(quantity , 0) AS
      quantity
FROM STORE
LEFT JOIN HAS_IN_STOCK
ON STORE.s_id = HAS_IN_STOCK.s_id AND HAS_IN_STOCK.p_id = 3
ORDER BY STORE.s_id ASC;
```

(c)

```
db2 =>          SELECT STORE.s_id, s_address, COALESCE(quantity, 0) AS quantity
          FROM STORE
          LEFT JOIN HAS_IN_STOCK
            ON STORE.s_id = HAS_IN_STOCK.s_id AND HAS_IN_STOCK.p_id = 3
          ORDER BY STORE.s_id ASC;
db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) =>
S_ID      S_ADDRESS      QUANTITY
-----
1 9301 Leopard St      0
2 456 Oakwood Dr      180
3 789 Maple Ave      0
4 321 Pine St      195
5 654 Cedar Ln      0
6 987 Birch Rd      180
7 159 Elm St      205
8 753 Walnut Blvd      0

8 record(s) selected.
```

Figure 2: Query 1 result

Query 2

- (a) List the total amount of money spent by each customer in the store with id = 1.
Output should include the customer's email and the total amount of money spent.

(b)

```
SELECT CUSTOMER.email, COALESCE(SUM(amount), 0) AS
      total_amount
FROM CUSTOMER
LEFT JOIN ONLINE ON CUSTOMER.email = ONLINE.email
LEFT JOIN PURCHASE ON ONLINE.p_id = PURCHASE.p_id
LEFT JOIN STORE ON ONLINE.s_id = STORE.s_id
GROUP BY CUSTOMER.email
ORDER BY email ASC;
```

(c)

```
db2 =>          SELECT CUSTOMER.email, COALESCE(SUM(amount), 0) AS total_amount
          FROM CUSTOMER
          LEFT JOIN ONLINE ON CUSTOMER.email = ONLINE.email
          LEFT JOIN PURCHASE ON ONLINE.p_id = PURCHASE.p_id
          LEFT JOIN STORE ON ONLINE.s_id = STORE.s_id
          GROUP BY CUSTOMER.email
          ORDER BY email ASC;
db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) =>
EMAIL      TOTAL_AMOUNT
-----
david.miller@example.com      238.87
emily.jones@example.com      66.46
james.white@example.com      0.00
jane.smith@example.com      0.00
john.doe@example.com      0.00
karen.martin@example.com      0.00
lisa.brown@example.com      66.45
mark.johnson@example.com      0.00
michael.wilson@example.com      88.45
susan.davis@example.com      0.00

10 record(s) selected.
```

Figure 3: Query 2 result

Query 3

- (a) List the id of the biggest in-store purchase made by each store.
Output should include the store id, address and the purchase amount.

(b)

```

SELECT STORE.s_id , s_address , COALESCE(MAX(amount), 0) AS
    max_purchase_amount
FROM STORE
LEFT JOIN EMPLOYEE ON STORE.s_id = EMPLOYEE.s_id
LEFT JOIN INSTORE ON EMPLOYEE.e_id = INSTORE.e_id
LEFT JOIN PURCHASE ON INSTORE.p_id = PURCHASE.p_id
GROUP BY STORE.s_id , s_address
ORDER BY STORE.s_id ASC;

```

- (c)

```

db2 =>      SELECT STORE.s_id, s_address, COALESCE(MAX(amount), 0) AS max_purchase_amount
FROM STORE
LEFT JOIN EMPLOYEE ON STORE.s_id = EMPLOYEE.s_id
LEFT JOIN INSTORE ON EMPLOYEE.e_id = INSTORE.e_id
LEFT JOIN PURCHASE ON INSTORE.p_id = PURCHASE.p_id
GROUP BY STORE.s_id, s_address
ORDER BY STORE.s_id ASC;
db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) =>
S_ID      S_ADDRESS                                     MAX_PURCHASE_AMOUNT
-----
1 9301 Leopard St                                     44.47
2 456 Oakwood Dr                                     107.45
3 789 Maple Ave                                       90.95
4 321 Pine St                                         97.45
5 654 Cedar Ln                                       105.93
6 987 Birch Rd                                       0.00
7 159 Elm St                                         0.00
8 753 Walnut Blvd                                    0.00

8 record(s) selected.

```

Figure 4: Query 3 result

Query 4

- (a) List the id and address of every store and the corresponding money ever spent at that store.

(b)

```

WITH TEMP_INSTORE AS (
SELECT STORE.s_id , s_address , COALESCE(SUM(amount), 0) AS
    total_amount
FROM STORE
LEFT JOIN EMPLOYEE ON STORE.s_id = EMPLOYEE.s_id
LEFT JOIN INSTORE ON EMPLOYEE.e_id = INSTORE.e_id
LEFT JOIN PURCHASE ON INSTORE.p_id = PURCHASE.p_id
GROUP BY STORE.s_id , s_address
),
TEMP_ONLINE AS (
SELECT STORE.s_id , s_address , COALESCE(SUM(amount), 0) AS
    total_amount
FROM STORE
LEFT JOIN ONLINE ON STORE.s_id = ONLINE.s_id
LEFT JOIN PURCHASE ON ONLINE.p_id = PURCHASE.p_id
GROUP BY STORE.s_id , s_address
)
SELECT TEMP_INSTORE.s_id , TEMP_INSTORE.s_address ,

```

```

TEMP_INSTORE.total_amount + TEMP_ONLINE.total_amount AS
    total_amount
FROM TEMP_INSTORE
LEFT JOIN TEMP_ONLINE ON TEMP_INSTORE.s_id = TEMP_ONLINE.s_id
ORDER BY TEMP_INSTORE.s_id ASC;

```

(c)

```

db2 => WITH TEMP_INSTORE AS (
SELECT STORE.s_id, s_address, COALESCE(SUM(amount), 0) AS total_amount
FROM STORE
LEFT JOIN EMPLOYEE ON STORE.s_id = EMPLOYEE.s_id
LEFT JOIN INSTORE ON EMPLOYEE.e_id = INSTORE.e_id
LEFT JOIN PURCHASE ON INSTORE.p_id = PURCHASE.p_id
GROUP BY STORE.s_id, s_address
),
TEMP_ONLINE AS (
SELECT STORE.s_id, s_address, COALESCE(SUM(amount), 0) AS total_amount
FROM STORE
LEFT JOIN ONLINE ON db2 (cont.) => STORE.s_id = ONLINE.s_id
LEFT JOIN PURCHASE ON ONLINE.p_id = PURCHASE.p_id
GROUP BY STORE.s_id, s_address
)
SELECT TEMP_INSTORE.s_id, TEMP_INSTORE.s_address, TEMP_INSTORE.total_amount + TEMP_ONLINE.total_amount AS total_amount
db2 (cont.) => FROM TEMP_INSTORE
db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) =
> db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => LEFT JOIN TEMP_ONLINE ON TEMP_INSTORE.s_id
= TEMP_ONLINE.s_id
ORDER BY TEMP_INSTORE.s_id ASC;
db2 (cont.) =>
S_ID S_ADDRESS TOTAL_AMOUNT
-----
1 9301 Leopard St 110.92
2 456 Oakwood Dr 195.90
3 789 Maple Ave 157.41
4 321 Pine St 279.86
5 654 Cedar Ln 162.39
6 987 Birch Rd 0.00
7 159 Elm St 0.00
8 753 Walnut Blvd 0.00

8 record(s) selected.

```

Figure 5: Query 4 result

Query 5

(a) List the Paint products that are in that are in maximum quantity in the store with id = 1. List the product id, name and quantity.

(b)

```

WITH TEMP AS (
    SELECT PRODUCT.p_id, p_name, COALESCE(quantity, 0) AS
        quantity
    FROM PAINT
    LEFT JOIN PRODUCT ON PRODUCT.p_id = PAINT.p_id
    LEFT JOIN HAS_IN_STOCK ON PRODUCT.p_id = HAS_IN_STOCK.p_id
    WHERE HAS_IN_STOCK.s_id = 1
    ORDER BY quantity DESC
)
SELECT p_id, p_name, quantity
FROM TEMP
WHERE quantity = (SELECT MAX(quantity) FROM TEMP);

```

(c)

```

db2 =>      WITH TEMP AS (
            SELECT PRODUCT.p_id, p_name, COALESCE(quantity, 0) AS quantity
            FROM PAINT
            LEFT JOIN PRODUCT ON PRODUCT.p_id = PAINT.p_id
            LEFT JOIN HAS_IN_STOCK ON PRODUCT.p_id = HAS_IN_STOCK.p_id
            WHERE HAS_IN_STOCK.s_id = 1
            ORDER BY quantity DESC
        )
        SELECT p_id, p_name, quantity
        FROM TEMP
        WHERE quantity = (SELECT MAX(quantity) FROM TEMP);
db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (
cont.) => db2 (cont.) =>
P_ID      P_NAME      QUANTITY
-----
4 Sunset Orange      300

1 record(s) selected.

```

Figure 6: Query 5 result

5 SQL Modifications

Mod 1

- (a) Temporarily increase the price of products that where manufactured by the manufacturer with name that ends with "Industries" by 10%.

(b)

```

UPDATE PRODUCT
SET unit_price = unit_price * 1.1
WHERE m_id IN (
    SELECT m_id
    FROM MANUFACTURER
    WHERE m_name LIKE '%Industries'
);

```

(c)

```

db2 =>      UPDATE PRODUCT
            SET unit_price = unit_price * 1.1
            WHERE m_id IN (
                SELECT m_id
                FROM MANUFACTURER
                WHERE m_name LIKE '%Industries'
            );
db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => DB20000I The SQL command completed
successfully.

```

Figure 7: Mod 1 result

Mod 2

- (a) Merge two manufacturers with the id1 = 1 and id2 = 2 into a new manufacturer with name "m_name1-m_name2".

(b)

```

-- Step 1: Insert a new manufacturer with the combined name
INSERT INTO MANUFACTURER (m_id, m_name)
SELECT MAX(m_id) + 1,
       (SELECT m_name FROM MANUFACTURER WHERE m_id = 1) || '-' ||
       (SELECT m_name FROM MANUFACTURER WHERE m_id = 2)
FROM MANUFACTURER;

-- Step 2: Update products to assign the new manufacturer (
with new m_id)
UPDATE PRODUCT
SET m_id = (SELECT MAX(m_id) FROM MANUFACTURER)
WHERE m_id IN (1, 2);

```

— Step 3: Delete the old manufacturers
DELETE FROM MANUFACTURER WHERE m_id IN (1, 2);

(c)

```
db2 => -- Step 1: Insert a new manufacturer with the combined name
INSERT INTO MANUFACTURER (m_id, m_name)
SELECT MAX(m_id) + 1,
       (SELECT m_name FROM MANUFACTURER WHERE m_id = 1) || '-' || (SELECT m_name FROM MANUFACTURER WHERE m_id = 2)
FROM MANUFACTURER;

-- Step 2: Update products to assign the new manufacturer (with new m_id)
UPDATE PRODUCT
SET m_id = (SELECT MAX(m_id) FROM MANUFACTURER)
WHERE m_id IN (1, 2);

-- Step 3: Delete the old manufacturers
DELETE FROM MANUFACTURER WHERE m_id IN (1, 2);
db2 => db2 (cont.) => db2 (cont.) => DB20000I The SQL command completed successfully.
db2 => db2 => db2 => db2 (cont.) => db2 (cont.) => DB20000I The SQL command completed successfully.
db2 => db2 => db2 => DB20000I The SQL command completed successfully.
```

Figure 8: Mod 2 result

6 Views

View 1

(a) The view lists the expensive purchases in descending order of amount, where the amount is greater than or equal to 80.

(b)

```
CREATE VIEW EXPENSIVE_PURCHASES AS
SELECT p_id, amount, p_date, p_time
FROM PURCHASE
WHERE amount >= 80;
```

```
db2 => CREATE VIEW EXPENSIVE_PURCHASES AS
SELECT p_id, amount, p_date, p_time
FROM PURCHASE
WHERE amount >= 80;db2 (cont.) => db2 (cont.) => db2 (cont.) =>
DB20000I The SQL command completed successfully.
```

(c)

```
db2 => SELECT * FROM EXPENSIVE_PURCHASES
ORDER BY amount DESC
FETCH FIRST 5 ROWS ONLY;db2 (cont.) => db2 (cont.) =>
```

P_ID	AMOUNT	P_DATE	P_TIME
9	182.41	03/09/2025	17:00:00
2	107.45	03/02/2025	11:20:00
5	105.93	03/05/2025	16:10:00
4	97.45	03/04/2025	15:45:00
3	90.95	03/03/2025	14:30:00

(d)

5 record(s) selected.

```
db2 => INSERT INTO EXPENSIVE_PURCHASES (p_id, amount, p_date, p_time)
VALUES (11, 50, '2025-03-11', '18:00:00');db2 (cont.) =>
DB20000I The SQL command completed successfully.
```

(e)

- (f) The insertion was completed successfully. However, it is interesting to note that since the amount was less than 80, it was not included in the view, and only inserted in the PURCHASE table. The explanation for this comes from the following DB manual explanation: For views that are not defined with WITH CHECK OPTION, you can insert rows that do not conform to the definition of the view. Those rows cannot appear in the view but are inserted into the base table of the view.

View 2

- (a) This view provides a summary of total amount spent on online purchases by each customer.

(b)

```
CREATE VIEW CUSTOMER_ONLINE_SPENDING AS
SELECT
    Customer.email ,
    Customer.c_name ,
    COALESCE(SUM(Purchase.amount), 0) AS total_spent
FROM Customer
JOIN Online ON Customer.email = Online.email
JOIN Purchase ON Online.p_id = Purchase.p_id
GROUP BY Customer.email , Customer.c_name;
```

```
db2 => CREATE VIEW CUSTOMER_ONLINE_SPENDING AS
SELECT
    Customer.email,
    Customer.c_name,
    COALESCE(SUM(Purchase.amount), 0) AS total_spent
FROM Customer
JOIN Online ON Customer.email = Online.email
JOIN Purchase ON Online.p_id = Purchase.p_id
GROUP BY Customer.email, Customer.c_name;db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) => db2 (cont.) =>
db2 (cont.) => db2 (cont.) =>
DB20000I The SQL command completed successfully.
```

(c)

```
db2 => SELECT * FROM CUSTOMER_ONLINE_SPENDING FETCH FIRST 5 ROWS ONLY;

EMAIL                                C_NAME                                TOTAL_SPENT
-----                                -
lisa.brown@example.com               Lisa Brown                            66.45
michael.wilson@example.com           Michael Wilson                        88.45
emily.jones@example.com               Emily Jones                           66.46
david.miller@example.com              David Miller                          238.87
```

(d)

```
db2 => INSERT INTO CUSTOMER_ONLINE_SPENDING (email, c_name, total_spent)
VALUES ('leon.petrinos@example.com', 'Leon Petrinos', 400.00);db2 (cont.) =>
DB21034E The command was processed as an SQL statement because it was not a
valid Command Line Processor command. During SQL processing it returned:
SQL0151N The column "TOTAL_SPENT" cannot be updated. SQLSTATE=42808
```

(e)

- (f) The insert failed with SQLSTATE 42808. The manual states: Individual columns in a view cannot be updated if the column is derived from an SQL function, an arithmetic expression, or a constant. The TOTAL_SPENT column is derived from the SUM function, so it cannot be updated.

7 Check Constraints

Check 1

(a)

(b)

(c)

(d)

Check 2

- (a)
- (b)
- (c)
- (d)

8 Creativity

9 Work Division

We had two meetings to discuss the project and the work division. We decided to divide the work as follows:

- Ahmed Tlili: questions 4, 5, 6
- Leon Petrinos: Relational Schema, question 3, 9
- Mathilde Peruzzo: ER Diagram, questions 7, 9