Project 3

Ahmed Tlili, Leon Petrinos, Mathilde Peruzzo

March 24, 2025

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)
  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)
  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, delivery_fee NOT NULL, email NOT NULL)
  FOREIGN KEY(p_id) REFERENCES Purchase(p_id)
  FOREIGN KEY(email) REFERENCES Customer(email)

Stored Procedure

(a) This stored procedure increases the discount of products that havn't been sold in the past year. The discount is increased by 10%, but the maximum discount is 50%. So a product that already has a 45% discount will get a 5% discout. The input parameter is the maximum discount.

(b)

```
CREATE OR REPLACE PROCEDURE DiscountInactiveProducts(IN
    max_discount INT)
BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE current_pid INT;
    DECLARE current_discount INT;

    DECLARE product_cursor CURSOR FOR
        SELECT p_id, COALESCE(discount_pourcentage, 0)
        FROM Product
        WHERE COALESCE(discount_pourcentage, 0) < max_discount;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    OPEN product_cursor;

    FETCH product_cursor INTO current_pid, current_discount;

    WHILE done = 0 DO

        IF NOT EXISTS (
            SELECT 1
            FROM Contains_purchase cp
            JOIN Purchase pur
            ON cp.p_id = pur.p_id
            WHERE cp.product_id = current_pid
                AND pur.p_date >= CURRENT DATE - 6 MONTHS
        ) THEN
            IF (current_discount + 10 > max_discount) THEN
                SET current_discount = max_discount;
            ELSE
                SET current_discount = current_discount + 10;
            END IF;

            UPDATE Product
            SET discount_pourcentage = current_discount
            WHERE p_id = current_pid;
        END IF;

        FETCH product_cursor INTO current_pid, current_discount;

    END WHILE;

    CLOSE product_cursor;
END
```

(c) d

Application Program

Indexing

Index 1

(a)
```
db2 => CREATE INDEX clustered_purchase_idx ON Purchase(p_date) CLUSTER;
DB20000I  The SQL command completed successfully.
```

(b) A clustered index on purchase date in the Purchase table is beneficial because purchases are frequently analyed based on dates and date ranges. Thus, sorting the purchases by date allows for efficient range queries, making it faster to access data for accounting purposes. An example query that would benefit from this index is the following:

```
SELECT SUM(amount) AS total
FROM Purchase
WHERE p_date >= '01/01/2025' AND p_date <= '12/31/2025';
```

This above query computes the total revenue for the year 2025. With this clustered index, the database can quickly locate the first matching row, and perform a sequential scan to retrieve all rows within the specified date range, without needing to follow the pointers of other data entries (value + rid), as in a non-clustered index, which could often leed to more IO.

Index 2

(a)
```
db2 => CREATE INDEX stock_idx ON Has_in_stock(s_id, quantity) CLUSTER;
DB20000I  The SQL command completed successfully.
```

(b) This index is on the s_id and quantity attribute of the Has_in_stock table. It is useful for this application to efficiently identify products that are running low in stock in a specific store, which is crucial for inventory management. An example query that would benefit from this index is the following:

```
SELECT p_id
FROM Has_in_stock
WHERE s_id = 0 AND quantity < 5;
```

This query identifies all products of a particular store where the quantity of the product is very limited. The fact that it is a clustered index, again, allows for efficient range queries, making it faster to access data for inventory management purposes. It also makes sense to use a clustered index because the other attributes of the table are id's, which are certainly not needed in a sorted order.

Visualisation

Vis 1

Vis 2

Creativity