MTG Data Store

By : Nahid Sekander

Problem Scenario

Data Store Development

- Built a fictional data store using an existing database to simulate an online marketplace for Magic: The Gathering (MTG) cards with limited inventory.

Customer Purchase Process

- Enabled customers to browse and purchase cards by selecting the card_id and specifying the desired quantity.

Purchase Validation

Conditions:

- Verified if the requested card_id exists in the data store.
- Checked if the available stock is sufficient to fulfill the requested quantity.

Feedback Mechanism

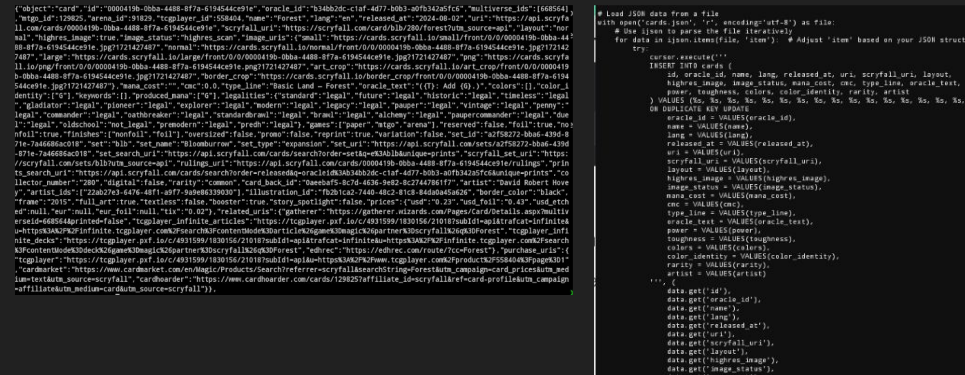
Return Message:

- Designed the system to return a success message upon meeting the validation criteria and completing the purchase
- Ensured the system provides clear error messages when the requested card is unavailable or stock levels are insufficient.

Building the data

- Discovered a dataset containing a comprehensive list of all MTG cards in CSV format.
- Developed a Python script to iterate through the CSV file and query a database, generating a complete list of all MTG cards.
- Filtered out duplicate entries, reducing the dataset from approximately 482,634 rows to around 33,047 rows.

JSON Example Data set



Pyhton Script to Populate DB

```
# Load JSON data from a file
with open('cards.json', 'r', encoding='utf-8') as file:
    # Use ijson to parse the file iteratively
    for data in ijson.items(file, 'item'):  # Adjust 'item' based on your JSON structure
        try:
            cursor.execute('''
            INSERT INTO cards (
                id, oracle_id, name, lang, released_at, uri, scryfall_uri, layout,
                highres_image, image_status, mana_cost, cmc, type_line, oracle_text,
                power, toughness, colors, color_identity, rarity, artist
            ) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
            ON DUPLICATE KEY UPDATE
                oracle_id = VALUES(oracle_id),
                name = VALUES(name),
                lang = VALUES(lang),
                released_at = VALUES(released_at),
                uri = VALUES(uri),
                scryfall_uri = VALUES(scryfall_uri),
                layout = VALUES(layout),
                highres_image = VALUES(highres_image),
                image_status = VALUES(image_status),
                mana_cost = VALUES(mana_cost),
                cmc = VALUES(cmc),
                type_line = VALUES(type_line),
                oracle_text = VALUES(oracle_text),
                power = VALUES(power),
                toughness = VALUES(toughness),
                colors = VALUES(colors),
                color_identity = VALUES(color_identity),
                rarity = VALUES(rarity),
                artist = VALUES(artist)
            ''', (
                data.get('id'),
                data.get('oracle_id'),
                data.get('name'),
                data.get('lang'),
                data.get('released_at'),
                data.get('uri'),
                data.get('scryfall_uri'),
                data.get('layout'),
                data.get('highres_image'),
                data.get('image_status'),
                data.get('mana_cost'),
                data.get('cmc'),
                data.get('type_line'),
                data.get('oracle_text'),
                data.get('power', None),  # Default to None if 'power' is missing
                data.get('toughness', None),  # Default to None if 'toughness' is missing
                ', '.join(data.get('colors', [])),  # Default to empty list if 'colors' is missing
                ', '.join(data.get('color_identity', [])),  # Default to empty list if 'color_identity' is missing
                data.get('rarity'),
                data.get('artist')
            ))
        except json.JSONDecodeError as e:
            print(f"Error decoding JSON: {e}")
        except MemoryError:
            print("MemoryError: Unable to load more data into memory. Exiting...")
            break
        except Error as e:
            print(f"Database error: {e}")

# Close the connection
if conn:
    # Commit changes and close the connection
    conn.commit()
    cursor.close()
    conn.close()

print("Data inserted successfully")
```

Full MTG Database Table

```
mariadb fnky@(none):mtg> describe unique_cards;
+----------------+--------------+------+-----+---------+-------+
| Field          | Type         | Null | Key | Default | Extra |
+----------------+--------------+------+-----+---------+-------+
| id             | varchar(36)  | NO   |     | <null>  |       |
| oracle_id      | varchar(36)  | YES  |     | <null>  |       |
| name           | varchar(255) | YES  | MUL | <null>  |       |
| lang           | varchar(10)  | YES  |     | <null>  |       |
| released_at    | date         | YES  |     | <null>  |       |
| uri            | varchar(255) | YES  |     | <null>  |       |
| scryfall_uri   | varchar(255) | YES  |     | <null>  |       |
| layout         | varchar(50)  | YES  |     | <null>  |       |
| highres_image  | tinyint(1)   | YES  |     | <null>  |       |
| image_status   | varchar(50)  | YES  |     | <null>  |       |
| mana_cost      | varchar(50)  | YES  |     | <null>  |       |
| cmc            | float        | YES  |     | <null>  |       |
| type_line      | varchar(255) | YES  |     | <null>  |       |
| oracle_text    | text         | YES  |     | <null>  |       |
| power          | varchar(10)  | YES  |     | <null>  |       |
| toughness      | varchar(10)  | YES  |     | <null>  |       |
| colors         | varchar(255) | YES  |     | <null>  |       |
| color_identity | varchar(255) | YES  |     | <null>  |       |
| rarity         | varchar(50)  | YES  |     | <null>  |       |
| artist         | varchar(255) | YES  |     | <null>  |       |
+----------------+--------------+------+-----+---------+-------+
```

Table Creation

Simplified MTG Card Table Creation:

- Created a new table named Full_MTG_Card_DB with the following fields:
  - id: Primary key with auto-increment.
  - name: VARCHAR(255) for storing the card's name.
  - description: Text field for card descriptions, including oracle text.

Data Migration:

- Extracted and inserted only the name and description/oracle text from the full MTG card list into the simplified mtg_cards table.

Inventory Population:

- Randomized and populated the quantity field for each card to simulate varying stock levels.

Customer Table:

- Created a customer table to store individual customer details, such as unique identifiers and contact information.

Checkout Table:

- Designed a checkout table to manage purchase transactions, including fields for transaction details and a status message to indicate the purchase's current state (e.g., pending, completed, or canceled).

```
INSERT INTO full_mtg_cards (Name, Description) SELECT name, oracle
                    -> _text FROM unique_cards;


CREATE INDEX idx_name ON full_mtg_cards(Name);
```

```
mariadb fnky@(none):mtg> describe `Full_MTG_Card_Database`;
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| Card_ID     | int(11)      | NO   | PRI | <null>  | auto_increment |
| Name        | varchar(255) | NO   | UNI | <null>  |                |
| Description | text         | NO   |     | <null>  |                |
+-------------+--------------+------+-----+---------+----------------+
```

```
source ~/COde/SQL/mtg_card_prices.sql;

CREATE TABLE IF NOT EXISTS combined_mtg_cards (
                    ->     card_id INT AUTO_INCREMENT PRIMARY KEY,
                    ->     name VARCHAR(255) NOT NULL UNIQUE,
                    ->     quantity INT NOT NULL,
                    ->     normal_price DOUBLE,
                    ->     foil_price DOUBLE
                    -> );
                    ->

INSERT IGNORE INTO combined_mtg_cards (name, normal_price, foil_price)
                    -> SELECT fmc.Name, mc.normal_price, mc.foil_price
                    -> FROM full_mtg_cards fmc
                    -> LEFT JOIN mtg_cards mc ON fmc.Name = mc.card_name;
```

```
UPDATE Data_Store_Available_Cards
SET quantity = IF(RAND() < 0.5, 0, FLOOR(RAND() * 31));
```

```
mariadb fnky@(none):mtg> describe `Customer_Information`;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| Customer_ID | int(11)      | NO   | PRI | <null>  |       |
| Name        | varchar(255) | NO   |     | <null>  |       |
| Email       | varchar(255) | NO   | UNI | <null>  |       |
| Address     | varchar(255) | NO   |     | <null>  |       |
+-------------+--------------+------+-----+---------+-------+
```

```
mariadb fnky@(none):mtg> describe `Customer_Checkout`;
+-------------+-------------+------+-----+---------+----------------+
| Field       | Type        | Null | Key | Default | Extra          |
+-------------+-------------+------+-----+---------+----------------+
| Checkout_ID | int(11)     | NO   | PRI | <null>  | auto_increment |
| Customer_ID | int(11)     | NO   | MUL | <null>  |                |
| Card_ID     | int(11)     | NO   | MUL | <null>  |                |
| Quantity    | int(11)     | NO   |     | <null>  |                |
| Status      | varchar(50) | NO   |     | <null>  |                |
+-------------+-------------+------+-----+---------+----------------+
```

View, Trigger and Functions

- View (Available_Inventory_View)
  - Purpose : To create a virtual table to display only the cards that have stock available
  - Includes card_id and quantiy
  - Filtes our rows where quantity is less than 1
- Trigger (check_stock_before_checkout)
  - Purpose : Ensures stock exists before a new row is inserted into the Customer_Checkout
  - Declare available_stock to store the current of the card being checked out
  - Checks if the stock is insufficient (Null or less than New.Quantity)
  - If insufficient, blocks the transaction with an error message
- Function (process_checkout)
  - Purpose :
    - Automates the process of recording a customer checkout
    - Ensures the checkout record is created with a status of “Pending”
    - Provides feedback to the caller about the status of the operation
  - Function Declaration
    - Input Parameters;
      - p_customer_id: The ID of the customer making the purchase
      - p_card_id : The ID of the card being purchased
      - p_quantity : The number of cards the customer wants to purchase
    - Return Type (VARCHAR(255));
      - The function will return a sucess message as a string
  - Variables : Declare a varaible msg to store the success message that will be
  - Insert Operations:
    - Inserts a new row into the Customer_Checkout table
      - Customer_ID: Set to the value of p_customer_id
      - Card_ID: Set to the value of p_card_id
      - Quantity: Set to the value of p_quantity
      - Status: Set to ‘Pending’ , indicating that the checkout process requires further validation
  - Set Success Message: Assigns a success message to the variable msg
  - Return the Message: Returns the value of msg to the caller, signalling that the checkout record has been created

```
CREATE VIEW Available_Inventory_View AS
                        -> SELECT
                        ->     card_id,
                        ->     quantity AS available_stock
                        -> FROM
                        ->     Data_Store_Available_Cards
                        -> WHERE
                        ->     quantity > 1;
                        ->
```

```
DELIMITER $$

CREATE TRIGGER check_stock_before_checkout
BEFORE INSERT ON Customer_Checkout
FOR EACH ROW
BEGIN
    DECLARE available_stock INT;

    -- Check available stock for the card being checked out
    SELECT quantity
    INTO available_stock
    FROM Data_Store_Available_Cards
    WHERE card_id = NEW.Card_ID;

    -- If the stock is not enough, block the checkout
    IF available_stock IS NULL OR available_stock < NEW.Quantity THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Insufficient stock for this card. Checkout b
locked.';
    END IF;

    -- If stock is sufficient, proceed and update the inventory
    UPDATE Data_Store_Available_Cards
    SET quantity = quantity - NEW.Quantity
    WHERE card_id = NEW.Card_ID;

END $$
```

```
DELIMITER $$

CREATE FUNCTION process_checkout(
    p_customer_id INT,        -- Customer ID
    p_card_id INT,            -- Card ID
    p_quantity INT            -- Quantity to purchase
)
RETURNS VARCHAR(255)
DETERMINISTIC
BEGIN
    DECLARE msg VARCHAR(255);

    -- Insert the record into the Customer_Checkout table with status 'Pending'
    INSERT INTO Customer_Checkout (Customer_ID, Card_ID, Quantity, Status)
    VALUES (p_customer_id, p_card_id, p_quantity, 'Pending');

    -- Set a success message
    SET msg = 'Checkout record created successfully, awaiting inventory validation.';

    -- Return the success message
    RETURN msg;
END $$

DELIMITER ;
```

ERD and Table Relationships

1. Customer Information → Customer Checkout (1:N):

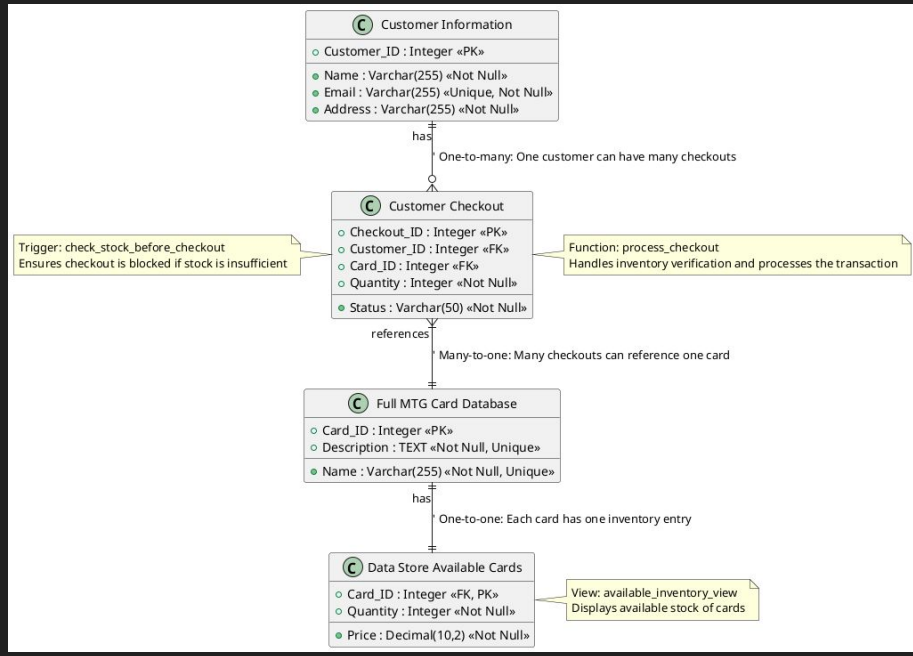
A Customer ID in the Customer Information table links to multiple entries in the Customer Checkout table via a foreign key. The Customer Checkout table tracks checkout-specific data, such as Checkout ID (Primary Key), Transaction Date, Total Amount, and Payment Method.

2. Customer Checkout → Full MTG Card Database (N:1):

Each entry in Customer Checkout can reference multiple cards, while the same card from the Full MTG Card Database can appear in multiple checkouts. This Many-to-Many relationship is managed via a junction table with fields like Checkout ID, Card ID, and Quantity Purchased.

3. Full MTG Card Database → Data Store Available Cards (1:1):

Each card in the Full MTG Card Database has a one-to-one mapping to an inventory record in the Data Store Available Cards table, which includes Stock Quantity, Restock Threshold, and Unit Price. The Card ID serves as the primary and foreign key between the two tables.



Thank You