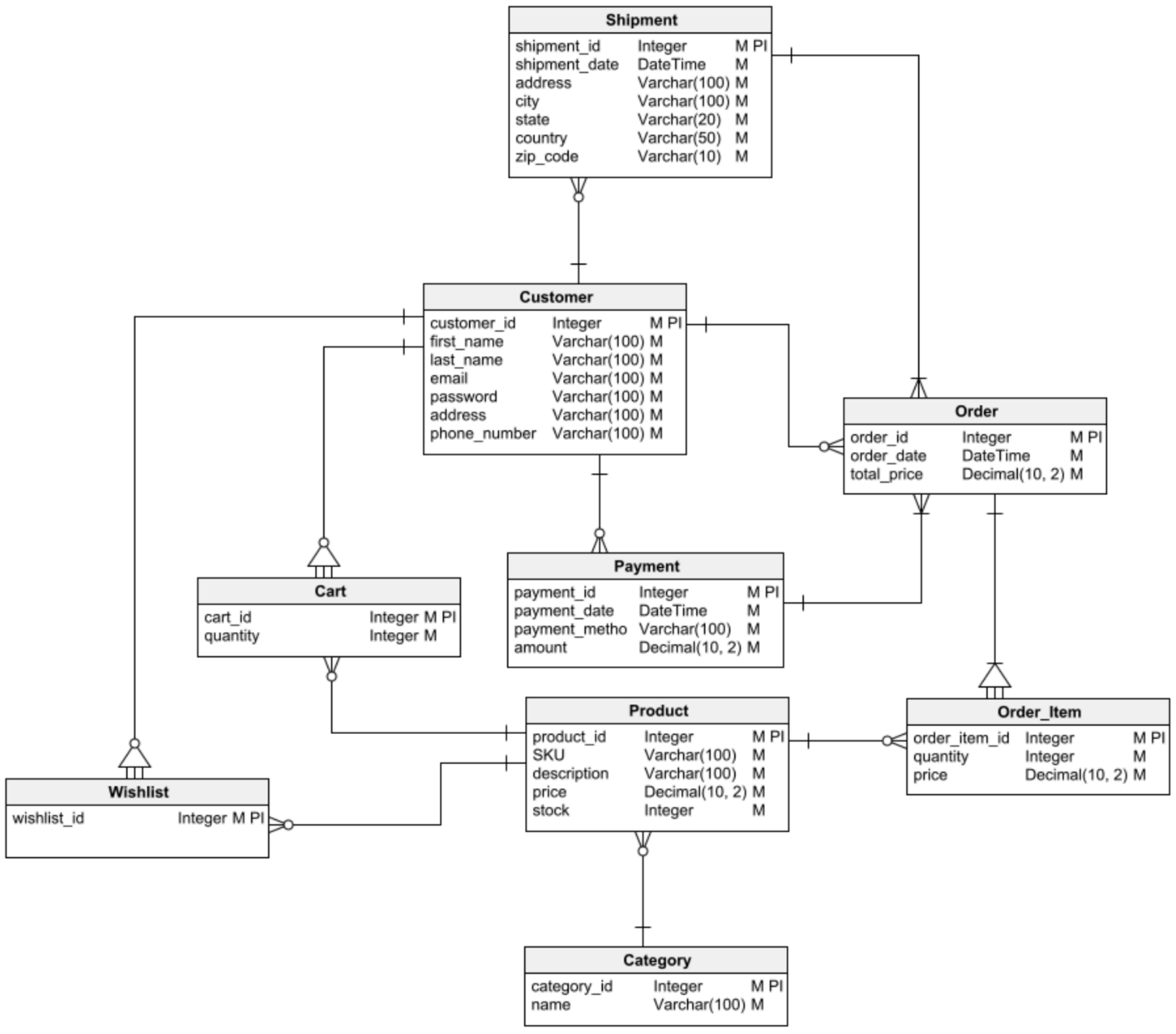
Day 08

Assignment 1: Analyze a given business scenario and create an ER diagram that includes entities, relationships, attributes, and cardinality. Ensure that the diagram reflects proper normalization up to the third normal form.

Solution :-



Assignment 2: Design a database schema for a library system, including tables, fields, and constraints like NOT NULL, UNIQUE, and CHECK. Include primary and foreign keys to establish relationships between tables.

Solution :-

CREATE TABLE BOOK(BOOK_ID NUMBER(5) PRIMARY KEY, BOOK_NAME VARCHAR2(50) NOT NULL, BOOK_QUANTITY NUMBER(5) CHECK(BOOK_QUANTITY>0));

CREATE TABLE LIBRARY(BOOK_ID NUMBER(5) REFERENCES BOOK(BOOK_ID), ORDER_NUMBER NUMBER(25), PRICE NUMBER(5) CHECK(PRICE>0));

Assignment 3: Explain the ACID properties of a transaction in your own words. Write SQL statements to simulate a transaction that includes locking and demonstrate different isolation levels to show concurrency control.

Solution :-

ACID properties are the golden rules that ensure the smooth operation of transactions in a database. Imagine a transaction as a complex recipe with multiple steps. ACID makes sure this recipe is followed correctly, and the final dish (data) is perfect every time.

1. **Atomicity:** All or nothing! This property guarantees that a transaction is treated as a single unit. Either all the steps within the transaction succeed, or none of them do.

2. **Consistency:** This property ensures the database transitions from one valid state to another.ACID guarantees your recipe maintains data integrity throughout the transaction.

3. **Isolation:** This property prevents transactions from interfering with each other. Isolation ensures each transaction appears to run alone, even if happening concurrently.

4. **Durability:** Once a transaction commits (finishes successfully), the changes become permanent. Durability guarantees that even if the power flickers (system crash), your delicious creation (data) is safely stored.

Here's an example using SQL to transfer money between two accounts:

```
START TRANSACTION;

-- Lock accounts to prevent conflicts (pessimistic locking)
SELECT * FROM accounts WHERE account_id = 1 FOR UPDATE;
SELECT * FROM accounts WHERE account_id = 2 FOR UPDATE;

-- Debit account 1
UPDATE accounts SET balance = balance - 100 WHERE account_id = 1;

-- Credit account 2
UPDATE accounts SET balance = balance + 100 WHERE account_id = 2;

COMMIT;
```

Assignment 4: Write SQL statements to CREATE a new database and tables that reflect the library schema you designed earlier. Use ALTER statements to modify the table structures and DROP statements to remove a redundant table.

Solution :-

—--> Creating two tables.

CREATE TABLE BOOK(BOOK_ID NUMBER(5) PRIMARY KEY, BOOK_NAME VARCHAR2(50) NOT NULL, BOOK_QUANTITY NUMBER(5) CHECK(BOOK_QUANTITY>0));

CREATE TABLE LIBRARY(BOOK_ID NUMBER(5) REFERENCES BOOK(BOOK_ID), ORDER_NUMBER NUMBER(25), PRICE NUMBER(5) CHECK(PRICE>0));

—-> Adding a row in BOOK table
ALTER TABLE BOOK ADD PAYMENT NUMBER(7);

—-> Dropping a table.
DROP TABLE LIBRARY;

Assignment 5: Demonstrate the creation of an index on a table and discuss how it improves query performance. Use a DROP INDEX statement to remove the index and analyze the impact on query execution.

Solution :-

Creating and Dropping an Index: Performance Impact

Let's consider a scenario where we have an e-commerce database table named Products with a column named **category**. We frequently query this table to find products belonging to specific categories.

Here's how indexing can improve performance:

1. **Creating an Index:**

We can create an index on the **category** column using the following SQL statement:

```
CREATE INDEX idx_category ON Products(category);
```

This creates a separate data structure that maps category values to their corresponding product data. It's like adding an index to a book - it allows for faster lookups.

2. **Impact on Query Performance:**

Now, consider a query that retrieves all products in the "Electronics" category:

```
SELECT * FROM Products WHERE category = 'Electronics';
```

Without an index, the database engine needs to scan every row in the **Products** table to find matching categories. This can be slow, especially for large tables.

With the index, the database can quickly locate entries for "Electronics" in the index and then efficiently retrieve the corresponding product data from the main table. This significantly reduces the search time.

3. **Dropping the Index:**

To demonstrate the impact of the index, let's drop it using the following statement:

DROP INDEX idx_category ON Products;

Now, if we re-run the same query:
SELECT * FROM Products WHERE category = 'Electronics';

The database will again need to scan the entire table, potentially leading to a noticeable performance decrease compared to when the index was present.

Assignment 6: Create a new database user with specific privileges using the CREATE USER and GRANT commands. Then, write a script to REVOKE certain privileges and DROP the user.

Solution :-
-- Create a new user with password
CREATE USER 'Rijin'@'localhost' IDENTIFIED BY 'Rijin12345';

-- Grant SELECT and INSERT privileges on the 'mydatabase' schema
GRANT SELECT, INSERT ON mydatabase.* TO 'Rijin'@'localhost';

-- Optionally, grant additional privileges as needed (e.g., UPDATE, DELETE)

-- List user privileges (optional)
SHOW GRANTS FOR 'Rijin'@'localhost';

-- Revoke INSERT privilege on the 'products' table within 'mydatabase'
REVOKE INSERT ON mydatabase.products FROM 'Rijin'@'localhost';

-- Drop the user
DROP USER 'Rijin'@'localhost';

Explanation:

1. **CREATE USER:** This statement creates a new user named **Rijin** with the specified password on the local machine (localhost).
2. **GRANT:** This statement grants the user **SELECT** and **INSERT** privileges on all tables within the **mydatabase** schema. You can modify this to include additional privileges (e.g., **UPDATE, DELETE**) or specific tables as needed.
3. **SHOW GRANTS (Optional):** This statement displays the current privileges granted to the user, allowing you to verify the configuration.
4. **REVOKE:** This statement revokes the **INSERT** privilege from the user for the **products** table within the **mydatabase** schema. This demonstrates revoking specific permissions.
5. **DROP USER:** This statement permanently removes the user named **Rijin** from the database.

Assignment 7: Prepare a series of SQL statements to INSERT new records into the library tables, UPDATE existing records with new information, and DELETE records based on specific criteria. Include BULK INSERT operations to load data from an external source.

Solutions :-

Tables (Assumptions):

- Books (book_id, title, author, ISBN, publication_year)
- Members (member_id, name, contact_info, membership_type)
- Loans (loan_id, book_id, member_id, loan_date, due_date, returned_date)

1. INSERT Statements:

a. **Single Record:**

```
INSERT INTO Books (title, author, ISBN, publication_year)
VALUES ('The Lord of the Rings', 'J.R.R. Tolkien', '9780261102694', 1954);
```

b. **Multiple Records:**

```
  INSERT INTO Members (name, contact_info, membership_type)
VALUES ('John Doe', 'johndoe@email.com', 'Standard'),
     ('Jane Smith', 'janesmith123@phone', 'Premium');
```

2. BULK INSERT (Database Specific):

Many databases offer BULK INSERT functionality to efficiently load data from external files (CSV, pdf, etc.).

```
-- Assuming a CSV file named 'new_books.csv'

BULK INSERT Books

FROM 'C:\data\new_books.pdf'
```

```
WITH (FIELDTERMINATOR = ',', ROWTERMINATOR = '\n');
```

3. UPDATE Statements:

a. **Update a book's publication year:**

```
UPDATE Books

SET publication_year = 2023

WHERE ISBN = '9780261102694';
```

b. **Update a member's contact information:**

```
UPDATE Members

SET contact_info = 'new_phone_number@phone'

WHERE member_id = 1;
```

4. DELETE Statements:

a. **Delete a book (be cautious!):**

```
DELETE FROM Books

WHERE ISBN = '9781234567890';
```

b. **Delete a loan record (assuming the book is returned):**

```
DELETE FROM Loans

WHERE loan_id = 123 AND returned_date IS NOT NULL;
```