Assignment-Day-8

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:

To create an ER diagram that includes entities, relationships, attributes, and cardinality, and ensures proper normalization up to the third normal form (3NF), follow these steps:

Here's a brief description of the necessary data:

Business Scenario:

1.Customers: Each customer has a unique customer ID, name, email address, and phone number.

2.Products: Each product has a unique product ID, name, description, and price.

3.Orders: Each order has a unique order ID, order date, and is associated with one customer. An order can contain multiple products.

4.Order Details: To track which products are in an order, there's a need for an order details entity. Each record here will have an order ID, product ID, quantity, and price at the time of order (to account for price changes).

4.Payments: Each payment has a unique payment ID, payment date, amount, and is associated with one order.

Step-by-Step ER Diagram Construction

1.Identify Entities and Attributes:

Customer: customer_id (PK), name, email, phone

Product: product_id (PK), name, description, price

Order: order_id (PK), order_date, customer_id (FK)

Order_Details: order_id (FK, PK), product_id (FK, PK), quantity, price

Payment: payment_id (PK), payment_date, amount, order_id (FK)

2.Define Relationships and Cardinality:

Customer to Order: One-to-Many (One customer can place many orders, but each order is placed by one customer)

Order to Order_Details: One-to-Many (One order can have multiple products, but each record in Order_Details refers to one order)

Product to Order_Details: Many-to-Many (One product can be in many orders, and one order can contain many products)

Order to Payment: One-to-Many (One order can have multiple payments, but each payment is for one order)

3.Normalization:

First Normal Form (1NF): Ensure each table has a primary key and each field contains only atomic values.

Second Normal Form (2NF): Ensure all non-key attributes are fully functional dependent on the primary key.

Third Normal Form (3NF): Ensure no transitive dependencies (i.e., non-key attributes depend only on the primary key).

Example ER Diagram:

Customer (customer_id (PK), name, email, phone)

Product (product_id (PK), name, description, price)

Order (order_id (PK), order_date, customer_id (FK))

Order_Details (order_id (FK, PK), product_id (FK, PK), quantity, price)

Payment (payment_id (PK), payment_date, amount, order_id (FK))

Relationships:

Customer (1) -- (M) Order

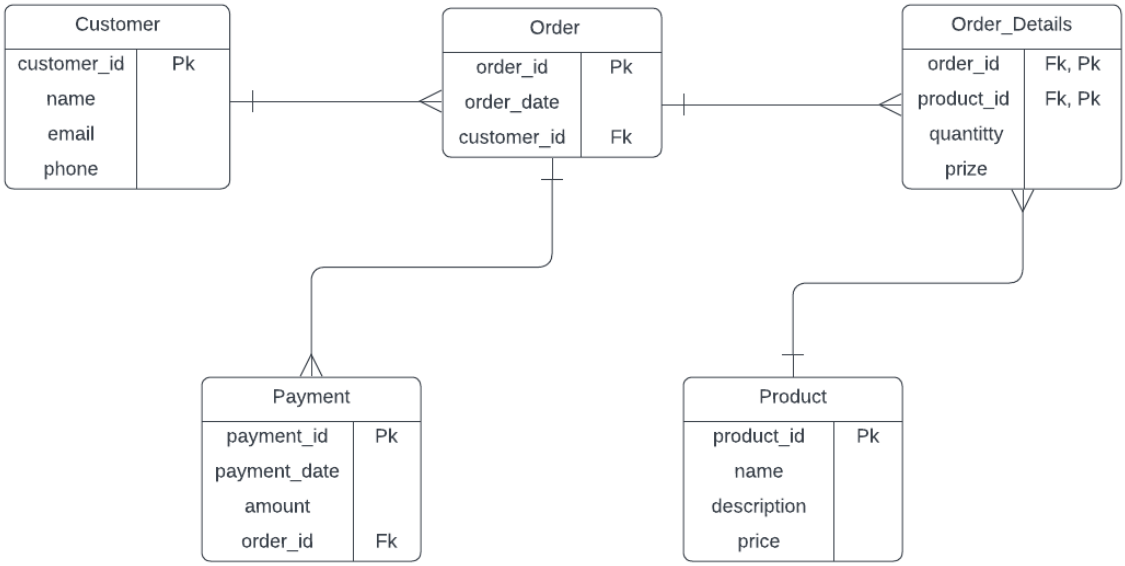
Order (1) -- (M) Order_Details

Product (1) -- (M) Order_Details

Order (1) -- (M) Payment

Create the ER Diagram

ER diagram: visualize the entities, relationships, and attributes.



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:

Tables and Fields

```
CREATE TABLE Authors (
    author_id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL
);


CREATE TABLE Books (
    book_id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    isbn VARCHAR(13) NOT NULL UNIQUE,
    publication_year YEAR NOT NULL CHECK (publication_year >= 1000 AND
publication_year <= 9999),
```

```
    author_id INT NOT NULL,
    FOREIGN KEY (author_id) REFERENCES Authors(author_id)
);

CREATE TABLE Members (
    member_id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    email VARCHAR(255) NOT NULL UNIQUE CHECK (email LIKE '%@%'),
    phone_number VARCHAR(15) UNIQUE
);

CREATE TABLE Loans (
    loan_id INT AUTO_INCREMENT PRIMARY KEY,
    book_id INT NOT NULL,
    member_id INT NOT NULL,
    loan_date DATE NOT NULL,
    return_date DATE CHECK (return_date >= loan_date),
    FOREIGN KEY (book_id) REFERENCES Books(book_id),
    FOREIGN KEY (member_id) REFERENCES Members(member_id)
);
```

Explanation of Constraints

Primary Key: Ensure each row in a table is unique.

NOT NULL: Ensures a column cannot have NULL values.

UNIQUE: Ensures all values in a column are different.

CHECK: Ensures all values in a column satisfy a specific condition.

Foreign Key: Establishes a relationship between tables, ensuring referential integrity.

This schema provides a structured way to manage a library system, with clear relationships between books, authors, members, and loans.

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 is an acronym that stands for Atomicity, Consistency, Isolation, and Durability. These properties ensure reliable processing of database transactions.

Atomicity: This means that a transaction is an all-or-nothing operation. Either all operations within the transaction are completed successfully, or none of them are. If any part of the transaction fails, the entire transaction is rolled back to its initial state.

Consistency: This ensures that a transaction brings the database from one valid state to another, maintaining database invariants. Any data written to the database must be valid according to all defined rules, including constraints, cascades, triggers, and any combination thereof.

Isolation: This property ensures that the operations of a transaction are isolated from those of other transactions. This means that the intermediate state of a transaction is invisible to other transactions. The degree of visibility can be controlled by various isolation levels.

Durability: Once a transaction has been committed, it will remain so, even in the event of a system failure. This means that the data is permanently stored in a non-volatile memory.

SQL Statements for Simulating a Transaction with Locking

Here's how to simulate a transaction with locking and demonstrate different isolation levels:

Simulating a Transaction

-- Start a transaction

START TRANSACTION;

-- Update a record

UPDATE Accounts SET balance = balance - 100 WHERE account_id = 1;

-- Introduce a delay to simulate long processing

-- **DO SLEEP(5);** -- This is pseudo-code to represent a delay

-- Update another record

UPDATE Accounts SET balance = balance + 100 WHERE account_id = 2;

-- Commit the transaction

COMMIT;

Demonstrating Isolation Levels for Concurrency Control

1. Read Uncommitted

In this isolation level, transactions can read data that has been modified by other transactions but not yet committed. This can lead to dirty reads.

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

START TRANSACTION;

-- Read data (possible dirty read)

SELECT * FROM Accounts WHERE account_id = 1;

-- Commit or rollback transaction

COMMIT;

2. Read Committed

In this isolation level, transactions can only read data that has been committed by other transactions, preventing dirty reads.

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

START TRANSACTION;

-- Read data (only committed data)

SELECT * FROM Accounts WHERE account_id = 1;

-- Commit or rollback transaction

COMMIT;

3. Repeatable Read

In this isolation level, all reads within the transaction will return the same data, preventing non-repeatable reads. However, phantom reads can still occur.

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

START TRANSACTION;

-- Read data (same data within the transaction)

SELECT * FROM Accounts WHERE account_id = 1;

-- Commit or rollback transaction

COMMIT;

4. Serializable

This is the highest isolation level, ensuring complete isolation. It prevents dirty reads, non-repeatable reads, and phantom reads by locking the data set.

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

START TRANSACTION;

-- Read data (fully isolated)

SELECT * FROM Accounts WHERE account_id = 1;

-- Commit or rollback transaction

COMMIT;

Explanation of Isolation Levels and Concurrency Control

Read Uncommitted: Allows dirty reads, offering the least isolation but the highest performance.

Read Committed: Prevents dirty reads, ensuring that only committed data is read.

Repeatable Read: Ensures that if a transaction reads a row, it will read the same value for that row if read again, preventing non-repeatable reads.

Serializable: Ensures complete isolation by locking the entire data set being read, which can lead to reduced performance but guarantees consistency.

These isolation levels help manage concurrency control, balancing between data integrity and system performance.

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 a New Database and Tables

First, we'll create a new database for the library system and then create the tables reflecting the schema designed earlier.

1. Create the Database

```
CREATE DATABASE Library_DB;

USE Library_DB;
```

2. Create the Tables

```
CREATE TABLE Authors (
    author_id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL
);
CREATE TABLE Books (
    book_id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    isbn VARCHAR(13) NOT NULL UNIQUE,
    publication_year YEAR NOT NULL CHECK (publication_year >= 1000 AND
publication_year <= 9999),
    author_id INT NOT NULL,
    FOREIGN KEY (author_id) REFERENCES Authors(author_id)
);
CREATE TABLE Members (
    member_id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    email VARCHAR(255) NOT NULL UNIQUE CHECK (email LIKE '%@%'),
    phone_number VARCHAR(15) UNIQUE
);


CREATE TABLE Loans (
    loan_id INT AUTO_INCREMENT PRIMARY KEY,
```

```
    book_id INT NOT NULL,
    member_id INT NOT NULL,
    loan_date DATE NOT NULL,
    return_date DATE CHECK (return_date >= loan_date),
    FOREIGN KEY (book_id) REFERENCES Books(book_id),
    FOREIGN KEY (member_id) REFERENCES Members(member_id)
);
```

Using ALTER Statements to Modify Table Structures

Here, we will add a new column to the Books table and modify an existing column in the Members table.

1. Add a New Column to the Books Table

```
ALTER TABLE Books
ADD COLUMN genre VARCHAR(50);
```

2. Modify an Existing Column in the Members Table

```
ALTER TABLE Members
MODIFY COLUMN phone_number VARCHAR(20);
```

Using DROP Statements to Remove a Redundant Table

Suppose we have a redundant table named Publishers that needs to be removed.

1. Drop the Publishers Table

```
DROP TABLE IF EXISTS Publishers;
```

Complete SQL Script

Combining all the above SQL statements, the complete script will look like this:

```
-- Create a new database
CREATE DATABASE Library_DB;
USE Library_DB;
-- Create the Authors table
CREATE TABLE Authors (
    author_id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL
```

```
);
-- Create the Books table
CREATE TABLE Books (
    book_id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    isbn VARCHAR(13) NOT NULL UNIQUE,
    publication_year YEAR NOT NULL CHECK (publication_year >= 1000 AND
publication_year <= 9999),
    author_id INT NOT NULL,
    FOREIGN KEY (author_id) REFERENCES Authors(author_id)
);
-- Create the Members table
CREATE TABLE Members (
    member_id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    email VARCHAR(255) NOT NULL UNIQUE CHECK (email LIKE '%@%'),
    phone_number VARCHAR(15) UNIQUE
);
-- Create the Loans table
CREATE TABLE Loans (
    loan_id INT AUTO_INCREMENT PRIMARY KEY,
    book_id INT NOT NULL,
    member_id INT NOT NULL,
    loan_date DATE NOT NULL,
    return_date DATE CHECK (return_date >= loan_date),
    FOREIGN KEY (book_id) REFERENCES Books(book_id),
    FOREIGN KEY (member_id) REFERENCES Members(member_id)
);
-- Add a new column to the Books table
```

ALTER TABLE Books

ADD COLUMN genre VARCHAR(50);

-- Modify an existing column in the Members table

ALTER TABLE Members

MODIFY COLUMN phone_number VARCHAR(20);

-- Drop the redundant Publishers table if it exists

DROP TABLE IF EXISTS Publishers;

This script creates a new database, defines the tables and their relationships, modifies table structures, and removes a redundant table.

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 an Index on a Table

Indexes are used to improve the speed of data retrieval operations on a database table at the cost of additional writes and storage space. Let's demonstrate the creation of an index on the Books table's title column.

1. Create an Index

CREATE INDEX idx_title ON Books(title);

How an Index Improves Query Performance

Indexes work similarly to an index in a book. Instead of scanning the entire table to find the rows that match a query, the database can use the index to locate the rows more quickly. This is especially useful for large tables. Let's compare query performance with and without the index.

Query Without Index

First, let's run a query to find books by title without using the index.

-- This is a query to find a book by title

SELECT * FROM Books WHERE title = 'Effective Java';

Without an index, the database will perform a full table scan to find the matching rows, which can be slow if the table contains a large number of records.

Query With Index

Now, let's run the same query after creating the index.

-- This query will use the index to find the book by title

SELECT * FROM Books WHERE title = 'Effective Java';

With the index in place, the database can quickly locate the row(s) that match the title 'Effective Java' using the idx_title index, significantly reducing the query execution time.

Dropping the Index

To remove the index, we use the DROP INDEX statement. Let's see how the removal of the index affects query performance.

2. Drop the Index

DROP INDEX idx_title ON Books;

Impact on Query Execution

After dropping the index, run the same query again:

-- Query to find a book by title after dropping the index

SELECT * FROM Books WHERE title = 'Effective Java';

Without the index, the database will revert to performing a full table scan to locate the matching rows, leading to longer query execution times, particularly for large datasets.

Summary

With Index

Query Execution: Fast, as the database uses the index to locate rows quickly.

Performance: Improved for read operations involving indexed columns.

Overhead: Slightly increased storage and slower write operations due to maintaining the index.

Without Index

Query Execution: Slower, as the database performs a full table scan to locate rows.

Performance: Decreased for read operations on large tables without the index.

Overhead: Reduced storage and faster write operations as no index maintenance is required.

Complete SQL Script

Here’s a complete script demonstrating the creation and dropping of an index:

-- Create the Books table

```
CREATE TABLE Books (
    book_id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    isbn VARCHAR(13) NOT NULL UNIQUE,
    publication_year YEAR NOT NULL CHECK (publication_year >= 1000 AND publication_year <= 9999),
    author_id INT NOT NULL,
    FOREIGN KEY (author_id) REFERENCES Authors(author_id)
);
-- Create an index on the title column
CREATE INDEX idx_title ON Books(title);
-- Query using the index
SELECT * FROM Books WHERE title = 'Effective Java';
-- Drop the index
DROP INDEX idx_title ON Books;
-- Query without the index
SELECT * FROM Books WHERE title = 'Effective Java';
```

This script demonstrates the creation of an index, its impact on query performance, and the effect of dropping the index.

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:

Creating a New Database User with Specific Privileges

Let's create a new database user and grant specific privileges to that user.

1. Create a New User

CREATE USER 'library_us'@'localhost' IDENTIFIED BY 'secure_password';

2. Grant Specific Privileges

We will grant the library_user privileges to select, insert, update, and delete data from the Library_DB database.

GRANT SELECT, INSERT, UPDATE, DELETE ON Library_DB.* TO 'library_user'@'localhost';

Script to Revoke Certain Privileges and Drop the User

Now, we will write a script to revoke certain privileges from the user and then drop the user.

1. Revoke Privileges

Let's revoke the DELETE privilege from library_user.

REVOKE DELETE ON Library_DB.* FROM 'library_user'@'localhost';

2. Drop the User

Finally, drop the user.

DROP USER 'library_user'@'localhost';

Complete SQL Script

Here is the complete script demonstrating the creation of a new user, granting specific privileges, revoking certain privileges, and dropping the user.

-- Create a new user

CREATE USER 'library_user'@'localhost' IDENTIFIED BY 'secure_password';

-- Grant specific privileges to the user

GRANT SELECT, INSERT, UPDATE, DELETE ON Library_DB.* TO 'library_user'@'localhost';

-- Revoke the DELETE privilege from the user

REVOKE DELETE ON Library_DB.* FROM 'library_user'@'localhost';

-- Drop the user

DROP USER 'library_user'@'localhost';

Explanation

CREATE USER: This command creates a new user with the specified username and password.

GRANT: This command grants specific privileges to the user on the specified database. In this case, we grant SELECT, INSERT, UPDATE, and DELETE privileges on all tables in the Library_DB database.

REVOKE: This command revokes specific privileges from the user. Here, we revoke the DELETE privilege from library_user.

DROP USER: This command deletes the user from the database.

Using these commands, you can manage database users and their privileges effectively.

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.

Solution:

-- Insert into Authors

INSERT INTO Authors (first_name, last_name) VALUES ('Joshua', 'Bloch');

INSERT INTO Authors (first_name, last_name) VALUES ('Herbert', 'Schildt');

-- Insert into Books

INSERT INTO Books (title, isbn, publication_year, author_id, genre)

VALUES ('Effective Java', '9780134685991', 2018, 1, 'Programming');

INSERT INTO Books (title, isbn, publication_year, author_id, genre)

VALUES ('Java: A Beginner's Guide', '9780071809252', 2014, 2, 'Programming');

-- Insert into Members

INSERT INTO Members (first_name, last_name, email, phone_number)

VALUES ('Alice', 'Smith', 'alice.smith@example.com', '555-123-4567');

INSERT INTO Members (first_name, last_name, email, phone_number)

VALUES ('Bob', 'Johnson', 'bob.johnson@example.com', '555-987-6543');

-- Insert into Loans

INSERT INTO Loans (book_id, member_id, loan_date, return_date)

VALUES (1, 1, '2024-05-05', '2024-05-19');

INSERT INTO Loans (book_id, member_id, loan_date, return_date)

VALUES (2, 2, '2024-05-12', '2024-05-26');

-- Update Author Name

UPDATE Authors

SET last_name = 'Gosling'

WHERE author_id = 2;

-- Update Book Genre

UPDATE Books

SET genre = 'Java Programming'

WHERE book_id = 1;

-- Delete a Member

DELETE FROM Members

WHERE email = 'bob.johnson@example.com';

-- Delete Old Loans

DELETE FROM Loans

WHERE return_date < '2024-05-05';

-- Create Authors for Bulk Insert (if they don't already exist)

INSERT INTO Authors (author_id, first_name, last_name)

VALUES (3, 'Kathy', 'Sierra'), (4, 'Bert', 'Bates');

-- Bulk Insert into Books

LOAD DATA INFILE '/path/to/java_books.csv'

INTO TABLE Books

FIELDS TERMINATED BY ','

LINES TERMINATED BY '\n'

IGNORE 1 LINES

(title, isbn, publication_year, author_id, genre);

This script demonstrates how to insert new records, update existing records, delete records based on specific criteria, and perform bulk insert operations.