Day_8

Date: 05-Aug-2024

Assignment 1: Analyse 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.

Ans:-

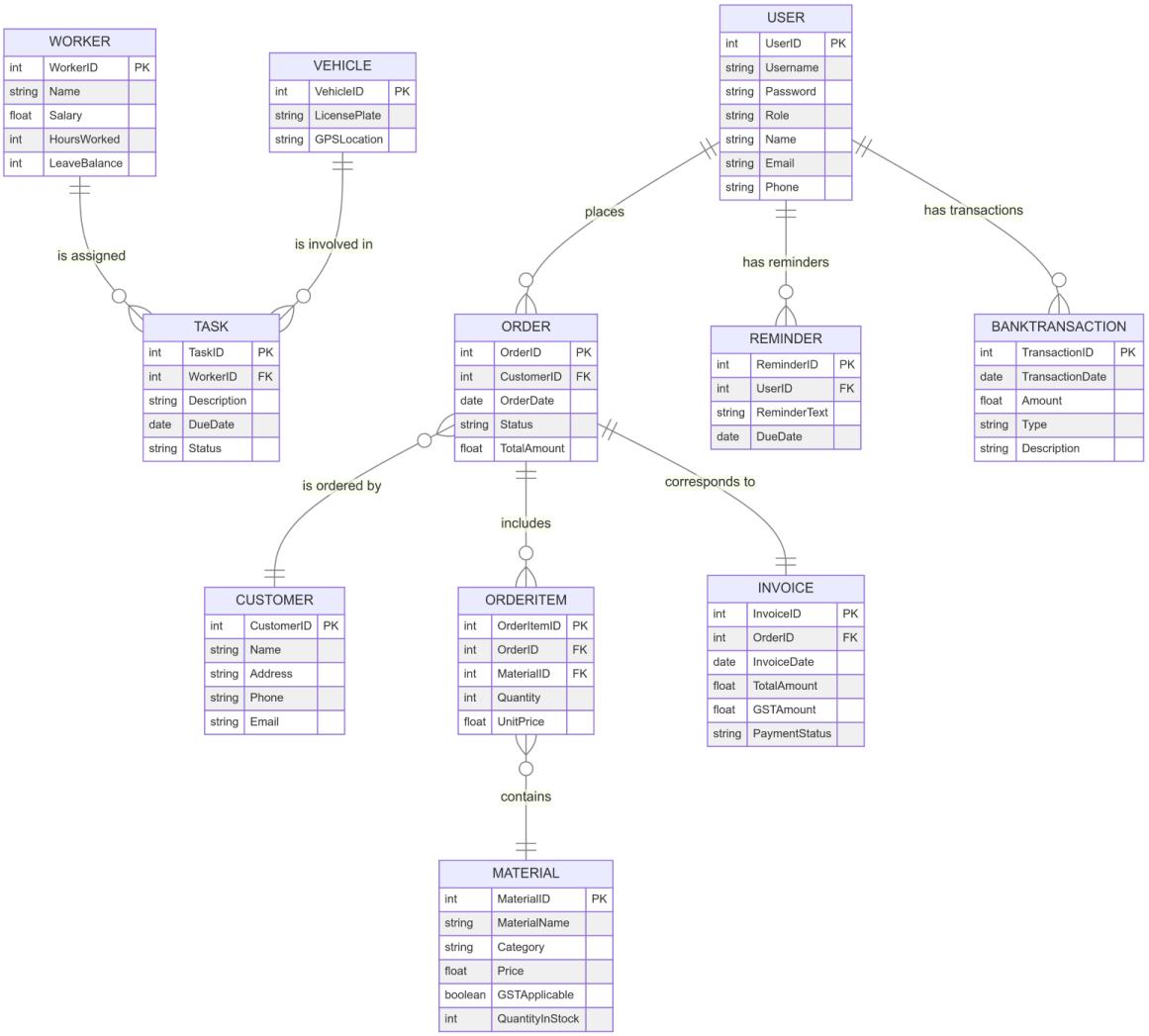
Here as a Business Scenario, I have taken same problem or project for this task i.e. Construction Material Management Software so here I have created solution based on that particular project.

For the project details ref. Day_4_Task in session file dated 30-July-2024

Here is short description about project:

The project is based on construction material selling business management. So here basically we have four roles Owner, Manager, Customer and Worker. The software supports and work accordingly to all these roles and manages inventory, orders, payment etc.

ER Diagram:



Entities and Attributes:

1. **USER**
   - **Attributes:** UserID (PK), Username, Password, Role, Name, Email, Phone

2. **CUSTOMER**
   - **Attributes:** CustomerID (PK), Name, Address, Phone, Email

3. **MATERIAL**
   - **Attributes:** MaterialID (PK), MaterialName, Category, Price, GSTApplicable, QuantityInStock

4. **ORDER**
    - **Attributes:** OrderID (PK), CustomerID (FK), OrderDate, Status, TotalAmount

5. **ORDERITEM**
    - **Attributes:** OrderItemID (PK), OrderID (FK), MaterialID (FK), Quantity, UnitPrice

6. **INVOICE**
    - **Attributes:** InvoiceID (PK), OrderID (FK), InvoiceDate, TotalAmount, GSTAmount, PaymentStatus

7. **WORKER**
    - **Attributes:** WorkerID (PK), Name, Salary, HoursWorked, LeaveBalance

8. **TASK**
    - **Attributes:** TaskID (PK), WorkerID (FK), Description, DueDate, Status

9. **VEHICLE**
    - **Attributes:** VehicleID (PK), LicensePlate, GPSLocation

10. **BANKTRANSACTION**
    - **Attributes:** TransactionID (PK), TransactionDate, Amount, Type, Description

11. **REMINDER**
    - **Attributes:** ReminderID (PK), UserID (FK), ReminderText, DueDate

Relationships and Cardinality:

i. **USER** places **ORDER**:
    a. **Cardinality:** One-to-Many (1:N)
    b. **Explanation:** One USER can place multiple ORDERS.

ii. **ORDER** is ordered by **CUSTOMER**:
    a. **Cardinality:** Many-to-One (N:1)
    b. **Explanation:** Multiple ORDERS can be made by one CUSTOMER.

iii. **ORDER** includes **ORDERITEM**:
    a. **Cardinality:** One-to-Many (1:N)
    b. **Explanation:** One ORDER can include multiple ORDERITEMs.

iv. **ORDERITEM** contains **MATERIAL**:
    a. **Cardinality:** Many-to-One (N:1)
    b. **Explanation:** Multiple ORDERITEMs can refer to one MATERIAL.

v. **ORDER** corresponds to **INVOICE**:
    a. **Cardinality:** One-to-One (1:1)
    b. **Explanation:** One ORDER corresponds to one INVOICE.

vi. **USER** has reminders **REMINDER**:
    a. **Cardinality:** One-to-Many (1:N)
    b. **Explanation:** One USER can have multiple REMINDERS.

vii. **WORKER** is assigned **TASK**:
    a. **Cardinality:** One-to-Many (1:N)
    b. **Explanation:** One WORKER can be assigned multiple TASKs.

viii. **VEHICLE** is involved in **TASK**:
    a. **Cardinality:** One-to-Many (1:N)
    b. **Explanation:** One VEHICLE can be involved in multiple TASKs.

ix. **USER** has transactions **BANKTRANSACTION**:
    a. **Cardinality:** One-to-Many (1:N)
    b. **Explanation:** One USER can have multiple BANKTRANSACTIONs.

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.

Ans:

Database Schema for Library System

Constraints, Tables and Fields:

1. **Author Table:**
    - Fields: AuthorID, FirstName, LastName, BirthDate
    - Constraints: AuthorID as primary key, FirstName and LastName not null

2. **Books Table:**
    - Fields: BookID, Title, AuthorID, PublishedYear, Genre
    - Constraints: BookID as primary key, Title not null, AuthorID as foreign key, PublishedYear with a check constraint

3. **Borrowers Table:**
    - Fields: BorrowerID, FirstName, LastName, MembershipDate, Email
    - Constraints: BorrowerID as primary key, FirstName, LastName, and Email not null, Email as unique, Email format check

4. **Checkout Table:**
    - Fields: CheckoutID, BookID, BorrowerID, CheckoutDate, ReturnDate
    - Constraints: CheckoutID as primary key, BookID and BorrowerID as foreign keys, CheckoutDate and ReturnDate not null

5. **Fines Table:**
    - Fields: FineID, BorrowerID, Amount, Paid
    - Constraints: FineID as primary key, BorrowerID as foreign key, Amount not null and positive, Paid with a check constraint

Description of Lib. Schema Diagram:

I. **Author Table:**
- AuthorID is set as the primary key and auto-incremented.
- FirstName and LastName are mandatory fields (NOT NULL).

II. **Books Table:**
- BookID is set as the primary key and auto-incremented.
- Title is a mandatory field (NOT NULL).
- AuthorID is a foreign key referencing Authors.
- PublishedYear has a check constraint to ensure it's a valid year.

III. **Borrowers Table:**
- BorrowerID is set as the primary key and auto-incremented.
- FirstName, LastName, and Email are mandatory fields (NOT NULL).
- Email must be unique (UNIQUE) and follow a specific format.

IV. **Checkout Table:**
- CheckoutID is set as the primary key and auto-incremented.
- BookID and BorrowerID are foreign keys referencing Books and Borrowers respectively.
- CheckoutDate and ReturnDate are mandatory fields (NOT NULL).

V. **Fines Table:**
- FineID is set as the primary key and auto-incremented.
- BorrowerID is a foreign key referencing Borrowers.
- Amount is a mandatory field (NOT NULL) and must be positive.
- Paid is a boolean field with a check constraint to ensure it's either 0 (false) or 1 (true).

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.

Ans:

❖ ACID- Properties:

- ACID properties are a set of characteristics that ensure data integrity in databases:

- **Atomicity**:
  - It make sure that all operations in a transaction are completed; if not, the transaction is rolled back entirely, providing an "all or nothing" approach.
  - Example: If you transfer money between two bank accounts, both the debit and credit operations must succeed, or neither should happen.

- **Consistency**:
  - Ensures that a transaction brings the database from one valid state to another, adhering to all defined rules and constraints.
  - Example: After a transaction, the total amount of money in both accounts should remain the same.

- **Isolation**:
  - Ensures that the operations of a transaction are not visible to other transactions until it is completed, maintaining a sequential execution appearance.
  - Example: One transaction's changes are not visible to other transactions until the transaction is completed.

- **Durability**:
  - Guarantees that once a transaction is committed, its changes are permanent, even in the event of a system failure.
  - Example: Once the money is transferred, it remains transferred even if the system crashes immediately after.

❖ SQL Statements to Simulate a Transaction:

- Create Tables:
  - Commands:

```
CREATE TABLE Accounts (
        AccountID INT PRIMARY KEY,
        Balance DECIMAL(10, 2) NOT NULL
);

INSERT INTO Accounts (AccountID, Balance) VALUES (1, 1000.00);
INSERT INTO Accounts (AccountID, Balance) VALUES (2, 2000.00);
```

- Transaction with Locking:
  - Commands:

```
-- Start the transaction
START TRANSACTION;

-- Lock the rows to prevent other transactions from accessing them
SELECT Balance FROM Accounts WHERE AccountID = 1 FOR
UPDATE;
SELECT Balance FROM Accounts WHERE AccountID = 2 FOR
UPDATE;

-- Perform the transfer
UPDATE Accounts SET Balance = Balance - 100 WHERE AccountID
= 1;
UPDATE Accounts SET Balance = Balance + 100 WHERE AccountID
= 2;

-- Commit the transaction
COMMIT;
```

❖ Isolation Levels:

➔ **READ UNCOMMITTED**:

- Allows dirty reads, where one transaction can see uncommitted changes made by another.
- Commands:

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
```

```
START TRANSACTION;
-- Read data that might be uncommitted
SELECT Balance FROM Accounts WHERE AccountID = 1;
COMMIT;
```

➔ **READ COMMITTED**:

- Prevents dirty reads; a transaction can only read committed changes.
- Commands:

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

START TRANSACTION;
-- Read only committed data
SELECT Balance FROM Accounts WHERE AccountID = 1;
COMMIT;
```

➔ **REPEATABLE READ**:

- Ensures that if a transaction reads data, the same data will be read again without any changes.
- Commands:

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

START TRANSACTION;
-- Read data and ensure it stays the same for this transaction
SELECT Balance FROM Accounts WHERE AccountID = 1;
-- Perform another read operation within the same transaction
SELECT Balance FROM Accounts WHERE AccountID = 1;
COMMIT;
```

➔ **SERIALIZABLE**:

- At the highest isolation level; transactions are executed in a way that they appear to be sequentially ordered.
- Commands:

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

START TRANSACTION;
-- Read data with the highest level of isolation
SELECT Balance FROM Accounts WHERE AccountID = 1;
-- Any new transactions that could affect this data will be blocked
until this one is complete
```

```
COMMIT;
```

➔ Description of Isolation Levels

- **READ UNCOMMITTED**:
  - Other transactions can see your changes before you commit them. Leads to dirty reads.
- **READ COMMITTED**:
  - Others see your changes only after you commit them. No dirty reads, but non-repeatable reads can happen.
- **REPEATABLE READ**:
  - Data read once will not change during the transaction. Prevents dirty and non-repeatable reads, but phantom reads can happen.
- **SERIALIZABLE**:
  - Transactions are completely isolated from each other, preventing dirty reads, non-repeatable reads, and phantom reads.