WESTMINSTER UNIVERSITY IN TASHKENT

COURSEWORK COVER PAGE

MODULE NAME: Database Systems Development

MODULE CODE: 5BUIS009C-n

LECTURER NAME: Dmitriy Pochitaev

WIUT STUDENT'S ID: 00016700

WORD COUNT: 2741 (update)

WESTMINSTER UNIVERSITY IN TASHKENT

Contents

1. Business Overview

A local music store engages in selling musical instruments, accessories, and providing music lessons. It also allows customers to rent instruments for specific durations. The store aims to implement a database system. This database must handle inventory (instruments), sales, rentals, customer information, lesson schedules, instructors' info, and supplier relationships. It should ensure data consistency, scalability, and ease of querying for reporting and decision-making.

Entities and Attributes

1. Customer

- **Attributes**:
  - ***CustomerID*** (Primary Key): A unique identifier for each customer.
  - ***Name***: The full name of the customer.
  - ***Address*** (Composite Attribute): Contains separate components for the customer's address:
    - ***Street***: The street name and number.
    - ***City***: The city name.
    - ***State***: The state or region.
    - ***ZipCode***: The postal code.
    - ***Country***: The country name.
  - ***Email***: Contact email address.
  - ***Phone*** (Multivalued Attribute): Customers may have multiple phone numbers.
- **Relationships**:
  - One-to-Many with ***Rental*** (CustomerID in ***Rental*** references ***CustomerID*** in ***Customer***).
  - One-to-Many with ***Sale*** (CustomerID in ***Sale*** references ***CustomerID*** in ***Customer***).
  - One-to-Many with ***Lesson*** (CustomerID in ***Lesson*** references ***CustomerID*** in ***Customer***).

2. Instrument

- **Attributes**:
  - ***InstrumentID*** (Primary Key): A unique identifier for each instrument.
  - ***InstrumentName***: The name of the instrument (e.g., Guitar, Piano).
  - ***Category***: Category of the instrument (e.g., String, Percussion).
  - ***SupplierID (Foreign Key)***: A unique identifier for each supplier.
- **Relationships**:
  - One-to-Many with ***Rental*** (InstrumentID in ***Rental*** references ***InstrumentID*** in ***Instrument***).
  - One-to-Many with ***Sale*** (InstrumentID in ***Sale*** references ***InstrumentID*** in ***Instrument***).
  - Many-to-Many with ***Supplier*** through ***SupplierInstrument*** (InstrumentID in ***SupplierInstrument*** references ***InstrumentID*** in ***Instrument***).

3. Rental

- **Attributes**:
  - ***RentalID*** (Primary Key): A unique identifier for each rental record.
  - ***CustomerID*** (Foreign Key): A unique identifier for each customer.
  - ***InstrumentID*** (Foreign Key): A unique identifier for each instrument.
  - ***RentalDate***: Date when the rental begins.
  - ***ReturnDate***: Date when the rental ends.
- **Relationships**:
  - ***CustomerID*** (Foreign Key): Links to the ***Customer*** entity.
  - ***InstrumentID*** (Foreign Key): Links to the ***Instrument*** entity.

4. Sale

- **Attributes**:
  - ***SalesID*** (Primary Key): A unique identifier for each sale.
  - ***CustomerID*** (Foreign Key): A unique identifier for each customer.
  - ***InstrumentID*** (Foreign Key): A unique identifier for each instrument.
  - ***SaleDate***: Date of the sale.
  - ***Price***: Price of the instrument.
- **Relationships**:
  - ***CustomerID*** (Foreign Key): Links to the ***Customer*** entity.
  - ***InstrumentID*** (Foreign Key): Links to the ***Instrument*** entity.

5. Lesson

- **Attributes**:
  - ***LessonID*** (Primary Key): A unique identifier for each lesson.
  - ***CustomerID*** (Foreign Key): A unique identifier for each customer.
  - **InstructorID** (Foreign Key): A unique identifier for each instructor.
  - **LessonDate**: Schedule of the lesson.
  - ***Topic***: Topic of the lesson.
- **Relationships**:
  - ***InstructorID*** (Foreign Key): Links to the ***Instructor*** entity.
  - ***CustomerID*** (Foreign Key): Links to the ***Customer*** entity.

6. Instructor

- **Attributes**:
  - ***InstructorID*** (Primary Key): A unique identifier for each instructor.
  - ***FirstName***: The first name of the instructor.
  - ***LastName***: The last name of the instructor.
  - ***Email***: The email address of the instructor.
  - ***PhoneNumber***: The phone number of the instructor.
- **Relationships**:
  - One-to-Many with ***InstructorExpertise*** (*InstructorID* in ***InstructorExpertise*** references ***InstructorID*** in ***Instructor***).

7. ExpertiseLevel

- **Attributes**:
  - ***ExpertiseLevelID*** (Primary Key): A unique identifier for each expertise level.
  - ***LevelName***: The name or description of the level of expertise.
- **Relationships**:
  - One-to-Many with ***InstructorExpertise*** (*ExpertiseLevelID* in ***InstructorExpertise*** references ***ExpertiseLevelID*** in ***ExpertiseLevel***).

8. InstrumentType

- **Attributes**:
  - ***InstrumentTypeID*** (Primary Key): A unique identifier for each instrument type.
  - ***InstrumentName***: The name of the instrument.
- **Relationships**:
  - One-to-Many with ***InstructorExpertise*** (*InstrumentTypeID* in ***InstructorExpertise*** references ***InstrumentTypeID*** in ***InstrumentType***).

9. InstructorExpertise

- **Attributes**:
  - ***InstructorExpertiseID*** (Primary Key): A unique identifier for each record of instructor expertise.
  - ***InstructorID*** (Foreign Key): References ***InstructorID*** in ***Instructor***.
  - ***InstrumentTypeID*** (Foreign Key): References ***InstrumentTypeID*** in ***InstrumentType***.
  - ***ExpertiseLevelID*** (Foreign Key): References ***ExpertiseLevelID*** in ***ExpertiseLevel***.
- **Relationships**:
  - Many-to-One with ***Instructor*** (InstructorID in ***InstructorExpertise*** references ***InstructorID*** in ***Instructor***).
  - Many-to-One with ***InstrumentType*** (InstrumentTypeID in ***InstructorExpertise*** references ***InstrumentTypeID*** in ***InstrumentType***).
  - Many-to-One with ***ExpertiseLevel*** (ExpertiseLevelID in ***InstructorExpertise*** references ***ExpertiseLevelID*** in ***ExpertiseLevel***).

10. Supplier

- **Attributes**:
  - ***SupplierID*** (Primary Key): A unique identifier for each supplier.
  - ***Name***: Supplier name.
  - ***ContactPerson***: Supplier contact details.
  - **PhoneNumber**: Contact phone number of the supplier.
  - **Email**: Email address of the supplier.
- **Relationships**:
  - Many-to-Many with ***Instrument*** through *SupplierInstrument* (SupplierID in ***SupplierInstrument*** references ***SupplierID*** in ***Supplier***).

11. Maintenance

- **Attributes**:
  - ***MaintenanceID*** (Primary Key): A unique identifier for each maintenance record.
  - ***InstrumentID*** (Foreign Key): A unique identifier for each instrument.
  - ***Date: Date of the maintenance.***
  - ***MaintenanceDetails***: Description of the maintenance performed.
  - **Cost**: Cost of the maintenance service.
- **Relationships**:
  - ***InstrumentID*** (Foreign Key): Links to the *Instrument* entity.

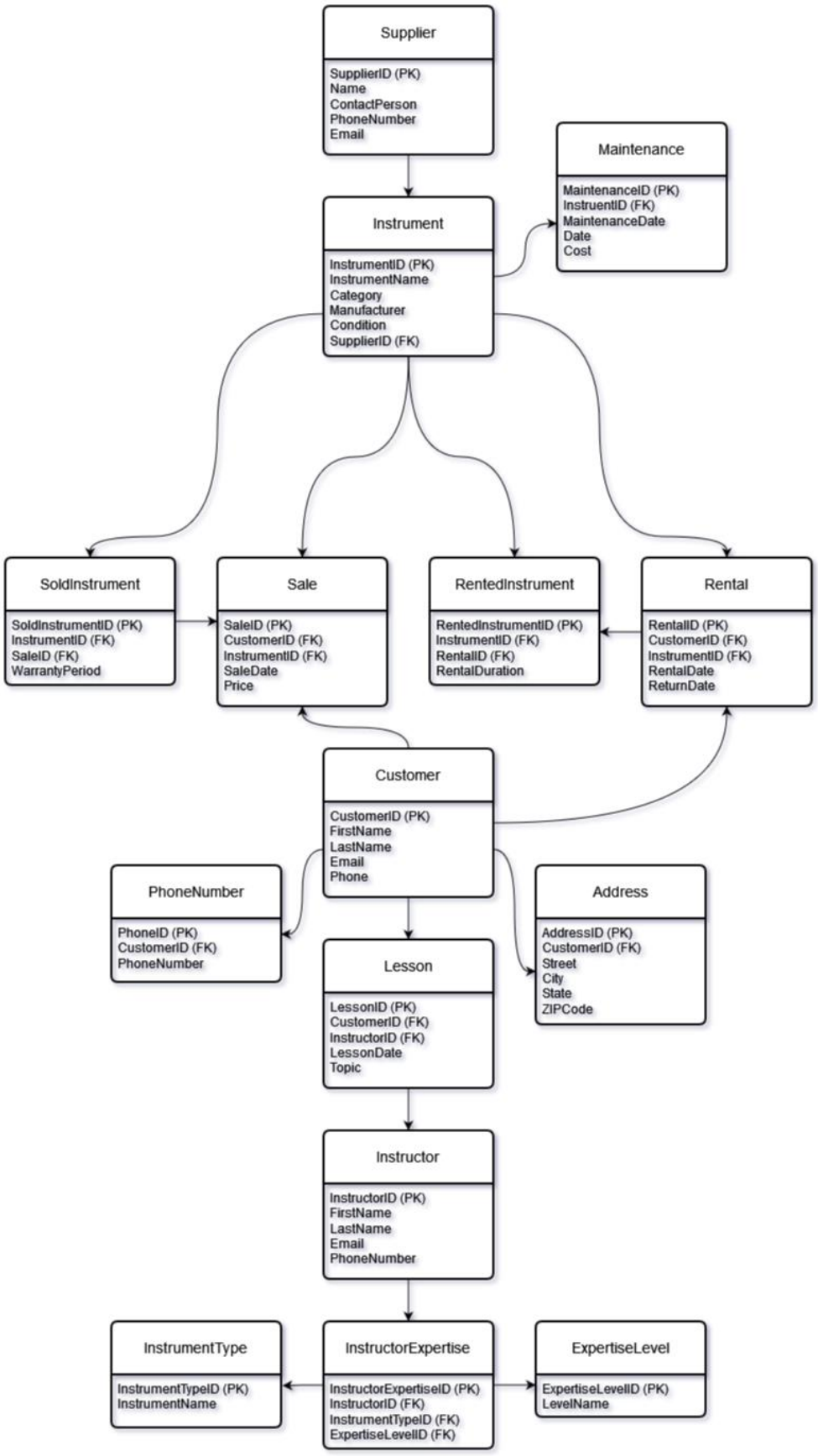
12. SoldInstrument

- **Attributes**:
  - ***SoldInstrumentID*** (Primary Key): A unique identifier for each sold instrument record.
  - ***InstrumentID*** (Foreign Key): References ***InstrumentID*** in ***Instrument***.
  - ***SaleID*** (Foreign Key): References ***SaleID*** in ***Sale***.
  - ***WarrantyPeriod***: The warranty period for the sold instrument.
- **Relationships**:
  - Many-to-One with ***Instrument*** (*InstrumentID* in ***SoldInstrument*** references ***InstrumentID*** in ***Instrument***).
  - Many-to-One with ***Sale*** (*SaleID* in ***SoldInstrument*** references ***SaleID*** in ***Sale***).

13. RentedInstrument

- **Attributes**:
  - ***RentedInstrumentID*** (Primary Key): A unique identifier for each rented instrument record.
  - ***InstrumentID*** (Foreign Key): References ***InstrumentID*** in ***Instrument***.
  - ***RentalID*** (Foreign Key): References ***RentalID*** in ***Rental***.
  - ***RentalDuration***: The duration for which the instrument is rented.
- **Relationships**:
  - Many-to-One with ***Instrument*** (*InstrumentID* in ***RentedInstrument*** references ***InstrumentID*** in ***Instrument***).
  - Many-to-One with ***Rental*** (*RentalID* in ***RentedInstrument*** references ***RentalID*** in ***Rental***).

2. Enhanced ER Diagram (ER)



ID: 00016700

The **Enhanced ER Diagram** includes:

- **Entities**: As defined above.
- **Superclass/Subclass**:
  - ***Instrument*** is a superclass with subclasses ***RentedInstrument*** and ***SoldInstrument***.
  - ***Rental*** and ***Sale*** are not considered subclasses, as they track transactions (actions) involving instruments.
- **Attributes**:
  - **Multivalued Attribute**: ***Customer.Phone***
  - **Composite Attribute**: ***Customer.Address***

***Note**: Multiplicity*

- **1:1 (One-to-One)**: Each record in Entity A is linked to only one record in Entity B, and vice versa.
- **1:M (One-to-Many)**: One record in Entity A can be linked to many records in Entity B, but each record in Entity B is linked to only one record in Entity A.
- **M:M (Many-to-Many)**: Many records in Entity A can be linked to many records in Entity B.

Multiplicity in Relationships:

Multiplicity defines how many instances of one entity (e.g., ***Customer***, ***Instrument***) can be linked to instances of another entity (e.g., ***Rental***, ***Sale***).

- ***Customer - Rental***: A customer can rent multiple instruments, but each rental record is associated with only one customer. *This is a 1:M relationship*.
- ***Customer - Sale***: A customer can purchase multiple instruments, but each sale record is associated with only one customer. *This is a 1:M relationship*.
- ***Customer - Lesson***: A customer can take multiple lessons, but each lesson is associated with one customer. *This is a 1:M relationship*.
- ***ExpertiseLevel - InstructorExpertise***: An expertise level can be associated with multiple instructors who share the same level of proficiency. Each record in ***InstructorExpertise*** points to one expertise level. *This is a 1:M relationship*.
- ***Instructor - InstructorExpertise***: An instructor can have multiple areas of expertise, meaning they can be associated with various instrument types and expertise levels. Each record in the ***InstructorExpertise*** table points to one instructor. *This is a 1:M relationship*.
- ***Instructor - Lesson***: An instructor can teach many lessons, but each lesson is taught by only one instructor. *This is a 1:M relationship*.
- ***Instrument - Maintenance***: An instrument can undergo many maintenance activities, but each maintenance record refers to one specific instrument. *This is a 1:M relationship*.
- ***Instrument - RentedInstrument***: An instrument can be rented multiple times, but each rental is associated with one instrument. *This is a 1:M relationship*.
- ***Instrument - SoldInstrument***: An instrument can be sold many times, but each sale is associated with one instrument. *This is a 1:M relationship*.

- ***InstrumentType - InstructorExpertise***: An instrument type can be associated with multiple instructors, each having varying expertise levels. Each record in ***InstructorExpertise*** links to one instrument type. *This is a 1:M relationship*.

Participation:

- **Partial Participation**:
  - The ***Instrument*** does not necessarily have to be sold or rented. It can exist without being part of either ***SoldInstrument*** or ***RentedInstrument***.

Disjointness:

- **Overlapping**:
  - An instrument can be both sold and rented simultaneously (overlap allowed).

3. Mapping EER Diagram to Relational Model

1. Superclass Attributes

Superclass *`Instrument`*

Shared Attributes: - **InstrumentID (PK)**: A unique identifier for each instrument. - **InstrumentName**: The name of the instrument (e.g., Piano, Guitar, etc.). - **Category**: The category of the instrument (e.g., string). - **Manufacturer**: The company that manufactured the instrument. - **Condition**: The condition of the instrument (default - new).

Note: Enforced at application layer. - **SupplierID (FK)**: A unique identifier of the supplier.

Justification: These attributes are **shared** across all instruments, regardless of whether they are sold or rented. Storing these attributes in a separate table ensures: - **Data Normalization**: Avoids duplication of common fields across subclasses. - **Consistency**: Any changes to common data (e.g., ***`Condition`*** or ***`Manufacturer`***) are updated in one place.

2. Subclass-Specific Attributes

Subclass 1: *`SoldInstrument`*** -** SoldInstrumentID (PK)**: A unique identifier for each sold instrument. - `InstrumentID (FK)` : A unique identifier for each instrument. - `SaleID (FK)` : A unique identifier for each instrument on sale. - WarrantyPeriod**: The warranty period of the instrument.

Subclass 2: *`RentedInstrument`*** - RentedInstrumentID (PK)**: - **InstrumentID (FK)**: A unique identifier for each instrument. - **RentalID (FK)**: A unique identifier for each instrument on rent. - **RentalDuration**: The date when the instrument is rented and should be returned.

Justification: These attributes are unique to the specific business processes of selling and renting instruments. Splitting them into subclasses ensures that: - Each table only contains relevant attributes, avoiding **NULL values** for unused fields (e.g., ***`WarrantyPeriod`*** in a rented instrument).

3. Why `Table per Subclass` Strategy was chosen?

Two alternative strategies, ***`Table per Hierarchy`*** and ***`Table per Concrete Class`***, were considered but deemed unsuitable.

The ***`Table per Hierarchy`*** approach involves storing all attributes in a single table with a discriminator column to differentiate subclasses. While this simplifies the schema, it leads to many NULL values for attributes not relevant to specific subclasses (e.g., ***`WarrantyPeriod`*** for rented instruments) and lacks support for enforcing subclass-specific constraints.

The ***`Table per Concrete Class`*** strategy, where each subclass is represented by its own table without a superclass table, results in significant data duplication for shared attributes like ***`InstrumentName`*** and ***`Manufacturer`***. This violates normalization principles, increases storage requirements, and complicates updates and queries.

In contrast, the ***`Table per Subclass`*** strategy avoids these issues by normalizing shared attributes in the *`Instrument`* table while maintaining separate tables for ***`SoldInstrument`*** and ***`RentedInstrument`***. This approach enforces constraints, avoids redundancy, and keeps the schema scalable and easy to maintain.

1. Reduces Redundancy and avoids sparse tables: - if all attributes were stored in a single table (e.g., ***Instrument***), many rows would have **NULL values** for attributes that do not apply (e.g., a sold instrument would have NULL values for ***RentalDuration***). - Splitting into separate subclass tables eliminated the redundancy.

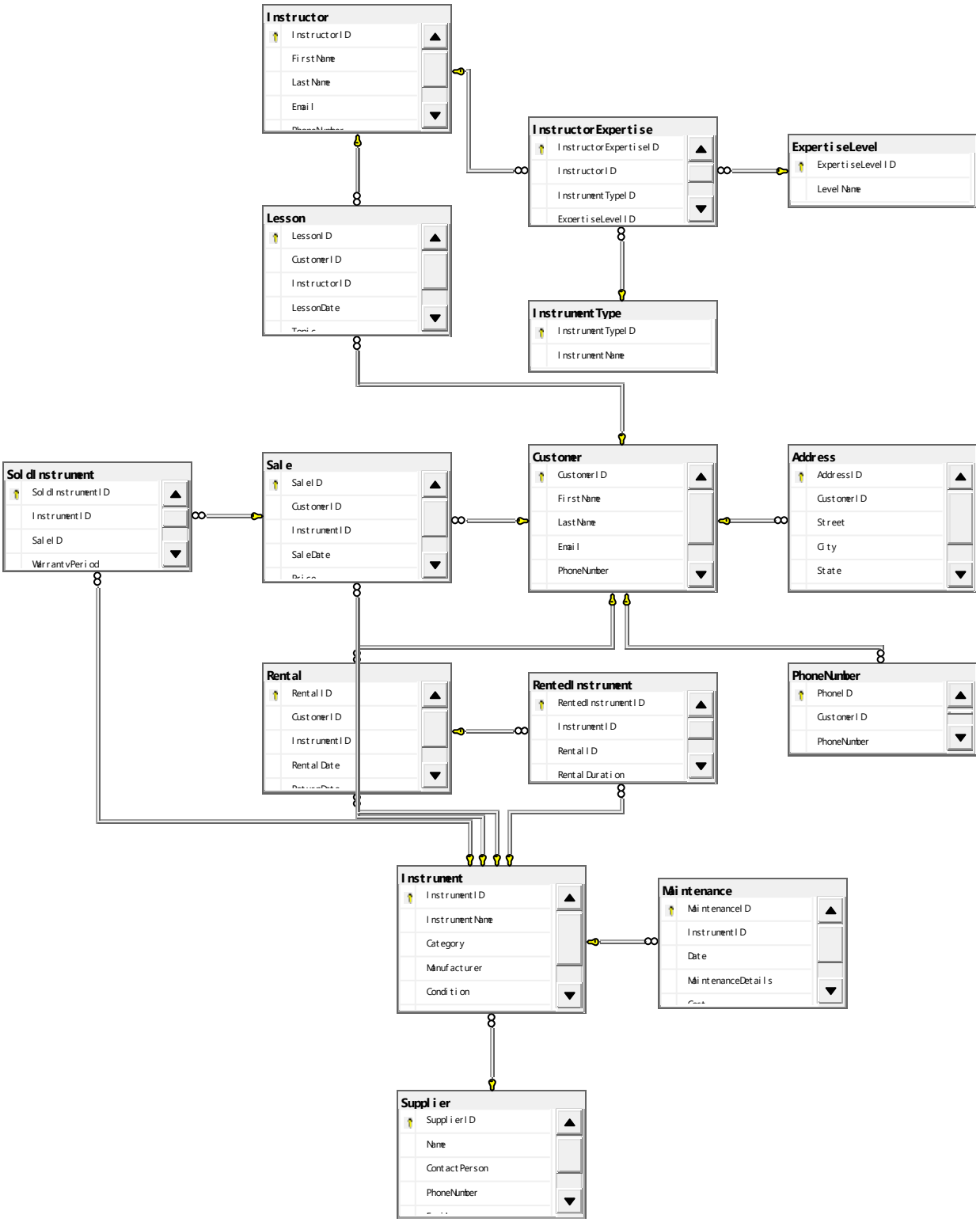
2. Enforces Constraints Unique to Each Subclass: - Constraints, such as ensuring that ***WarrantyPeriod*** is **NOT NULL** for sold instruments, can be applied at the table level. - This improves data integrity and reduces reliance on application-level validation.

3. Aligns with MS SQL Server's Relational Schema: - MS SQL Server's relational schema design supports primary and foreign key relationships, making it easy to implement the *Table per Subclass* strategy.

4. Implementation in Relational Schema

Tables and Relationships:

1. ***Instrument*** **Table (Superclass)**
   - Stores shared attributes
   - Acts as a **parent table** for subclasses.
   - **Primary Key**: ***InstrumentID***.
2. ***SoldInstrument*** **Table (Subclass)**
   - References ***Instrument*** via ***InstrumentID*** as a **foreign key**.
   - Contains attributes specific to sold instruments.
   - **Primary Key**: ***SoldInstrumentID***.
3. ***RentedInstrument*** **Table (Subclass)**
   - References ***Instrument*** via ***InstrumentID*** as a **foreign key**.
   - Contains attributes specific to rented instruments.
   - **Primary Key**: ***RentedInstrumentID***.



Note: SQL Note: Code is attached as a separate file.

1. ***Instument*** **Table**

```
CREATE TABLE Instrument (
    InstrumentID INT PRIMARY KEY IDENTITY(1,1), -- Unique identifier for
each instrument
    InstrumentName VARCHAR(100) NOT NULL, -- Name of the instrument
    Category VARCHAR(50), -- Category of the instrument (e.g., string, p
ercussion)
    Manufacturer VARCHAR(100), -- Manufacturer of the instrument
    Condition VARCHAR(50) DEFAULT 'New', -- Condition of the instrument
(default set to 'New')
    SupplierID INT NOT NULL, -- Foreign Key to Supplier table
    FOREIGN KEY (SupplierID) REFERENCES Supplier(SupplierID)
);
```

2. *SoldInstrument* **Table**

```
CREATE TABLE SoldInstrument (
    SoldInstrumentID INT PRIMARY KEY IDENTITY(1,1), -- Unique ID for sol
d instruments
    InstrumentID INT NOT NULL, -- Foreign Key to Instrument table
    SaleID INT NOT NULL, -- Foreign Key to Sale table
    WarrantyPeriod INT, -- Warranty period for the sold instrument in mo
nths
    FOREIGN KEY (InstrumentID) REFERENCES Instrument(InstrumentID),
    FOREIGN KEY (SaleID) REFERENCES Sale(SaleID)
);
```

3. *RentedInstrument* **Table**

```
CREATE TABLE RentedInstrument (
    RentedInstrumentID INT PRIMARY KEY IDENTITY(1,1), -- Unique ID for r
ented instruments
    InstrumentID INT NOT NULL, -- Foreign Key to Instrument table
    RentalID INT NOT NULL, -- Foreign Key to Rental table
    RentalDuration INT, -- Duration of the rental in days
    FOREIGN KEY (InstrumentID) REFERENCES Instrument(InstrumentID),
    FOREIGN KEY (RentalID) REFERENCES Rental(RentalID)
);
```

4. Validation and Normalization

Objective

This part validates the relational database schema to ensure it adheres to the Third Normal Form (3NF). Where necessary, functional dependencies (FDs) violating 3NF are introduced, analyzed, and resolved through normalization. The schema is updated and demonstrated using SQL scripts and example data.

1. Functional Dependencies and Violations

Initial Analysis

The original schema satisfies 3NF. However, to demonstrate normalization, an assumption was added:

Assumption:

- An instrument's category determines its manufacturer:
  `Category` → ***`Manufacturer`***

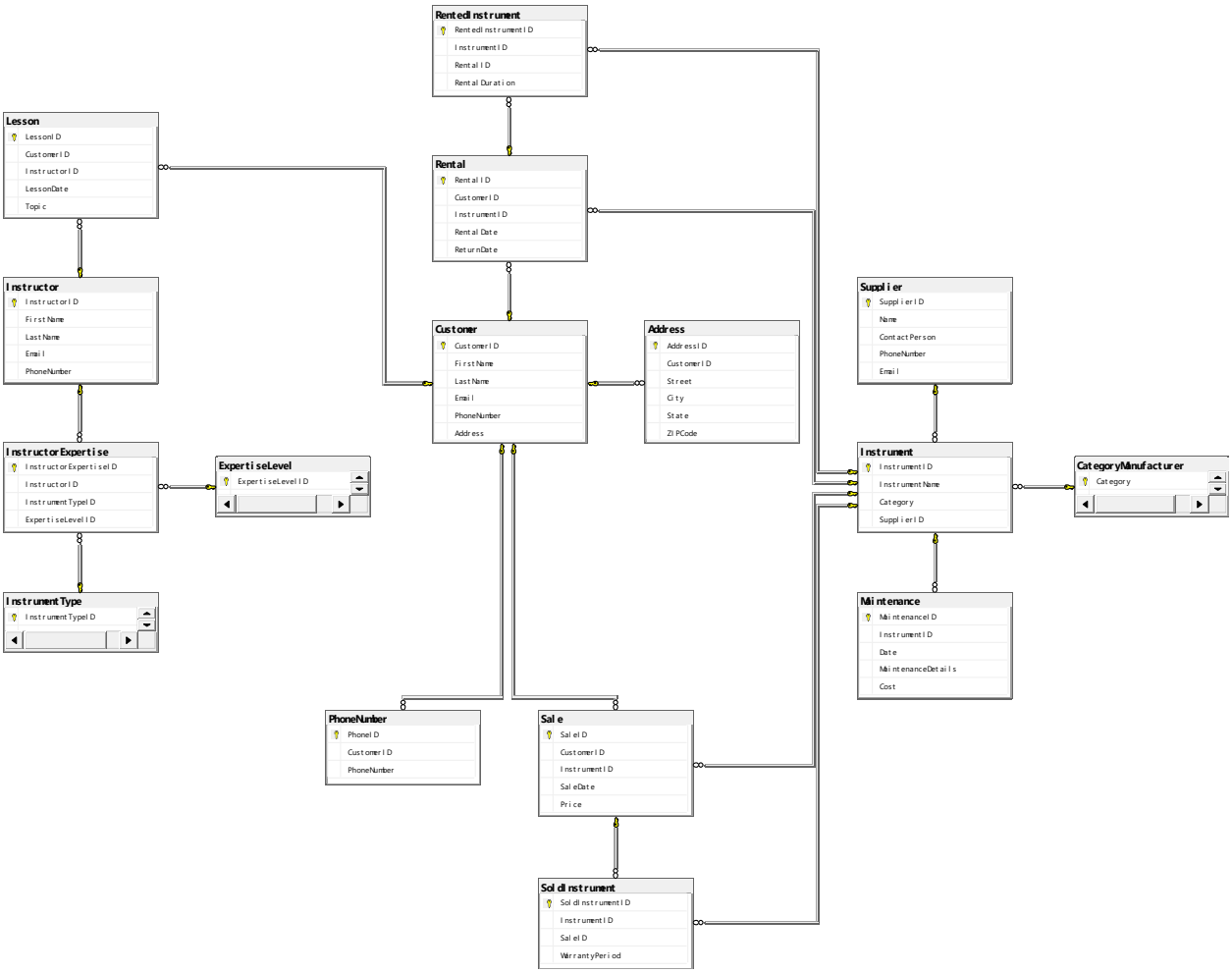
Impact of the New FD

This assumption introduces a **transitive dependency** in the ***`Instrument`*** **Table**:

- Primary key: ***`InstrumentID`***
- Dependencies:
  - ***`InstrumentID`*** → ***`Category`***
  - ***`Category`*** → ***`Manufacturer`***

Violation:
This creates a transitive dependency:
`InstrumentID` → ***`Category`*** → ***`Manufacturer`***

- **Reason for Violation:**
  - ***`Manufacturer`*** is dependent on *`Category`*, which is not part of the primary key (***`InstrumentID`***).
  - This violates 3NF, where all non-prime attributes must be directly dependent on the table's primary key.



2. Normalization Process

Decomposition to 3NF

To resolve the violation, the ***Instrument*** **Table** was decomposed into two tables:

1. **CategoryManufacturer Table**
   - Captures the functional dependency ***Category*** → ***Manufacturer***.
   - Eliminates redundancy in the ***Instrument*** **Table**.

Schema:

```
CREATE TABLE CategoryManufacturer (
    Category VARCHAR(50) PRIMARY KEY,
    Manufacturer VARCHAR(100) NOT NULL
);
```

2. **Modified Instrument Table**
   - Retains attributes directly dependent on the primary key (***InstrumentID***).

Schema:

```
CREATE TABLE Instrument (
    InstrumentID INT PRIMARY KEY,
    InstrumentName VARCHAR(100) NOT NULL,
    Category VARCHAR(50) NOT NULL,
    SupplierID INT NOT NULL,
    FOREIGN KEY (SupplierID) REFERENCES Supplier(SupplierID),
    FOREIGN KEY (Category) REFERENCES CategoryManufacturer(Category)
);
```

Steps to Normalize

1. Identified the transitive dependency: ***InstrumentID*** → ***Category*** → ***Manufacturer***.
2. Moved the dependency ***Category*** → ***Manufacturer*** to a new table (***CategoryManufacturer***).
3. Updated ***Instrument*** to reference ***CategoryManufacturer*** via the ***Category*** attribute.

3. Final Schema

CategoryManufacturer Table

Stores the FD ***Category*** → ***Manufacturer***.

```
CREATE TABLE CategoryManufacturer (
    Category VARCHAR(50) PRIMARY KEY,
    Manufacturer VARCHAR(100) NOT NULL
);
```

Instrument Table

References ***CategoryManufacturer*** for the ***Category*** attribute.

```
CREATE TABLE Instrument (
    InstrumentID INT PRIMARY KEY,
    InstrumentName VARCHAR(100) NOT NULL,
    Category VARCHAR(50) NOT NULL,
    SupplierID INT NOT NULL,
    FOREIGN KEY (SupplierID) REFERENCES Supplier(SupplierID),
    FOREIGN KEY (Category) REFERENCES CategoryManufacturer(Category)
);
```

4. Data Population

Populating CategoryManufacturer

```
INSERT INTO CategoryManufacturer (Category, Manufacturer)
VALUES
    ('String', 'Fender'),
    ('Percussion', 'Yamaha'),
    ('Keyboard', 'Steinway'),
    ('Wind', 'Yamaha'),
    ('Electronic', 'Roland')
```

Populating Instrument

```
INSERT INTO Instrument (InstrumentID, InstrumentName, Category, Supplier
ID)
```

```
VALUES
    (1, 'Guitar', 'String', 2),-- SupplierID 2 corresponds to Fender
    (2, 'Drum Set', 'Percussion', 1),-- SupplierID 1 corresponds to Yamaha
    (3, 'Piano', 'Keyboard', 3),-- SupplierID 3 corresponds to Steinway
    (4, 'Flute', 'Wind', 1);-- SupplierID 1 corresponds to Yamaha
    (5, 'Synthesizer', 'Electronic', 5),-- SupplierID 5 corresponds to Roland
    (6, 'Violin', 'String', 4);-- SupplierID 4 corresponds to Stradivarius
```

5. Verification

Join Query: Retrieve Instrument Details with Manufacturer

```
SELECT
    i.InstrumentID,
    i.InstrumentName,
    i.Category,
    cm.Manufacturer,
    i.SupplierID
FROM
    Instrument i
JOIN
    CategoryManufacturer cm ON i.Category = cm.Category;
```

Result:

InstrumentID	InstrumentName	Category	Manufacturer	SupplierID
1	Guitar	String	Fender	2
2	Drum Set	Percussion	Yamaha	1
3	Piano	Keyboard	Steinway	3
4	Flute	Wind	Yamaha	1
5	Synthesizer	Electronic	Roland	5
6	Violin	String	Fender	4

6. Explanation of Changes

Why These Changes Were Made

- To eliminate the transitive dependency ***InstrumentID*** → ***Category*** → ***Manufacturer***.
- To ensure all tables are in 3NF, with non-key attributes directly dependent on the primary key.

How the Schema Satisfies 3NF

1. **Instrument Table:**
   - All attributes (***InstrumentName***, ***Category***, ***SupplierID***) depend solely on the primary key (***InstrumentID***).
2. **CategoryManufacturer Table:**
   - The ***Manufacturer*** attribute depends solely on the primary key (***Category***).

Benefits of the Normalized Schema

- Eliminates redundancy by storing ***Manufacturer*** information in one place.

- Avoids anomalies during updates, deletions, or insertions.
- Maintains data integrity across related tables.

7. Conclusion

Validation and normalization of the relational schema to 3NF have been completed successfully. A functional dependency that violates 3NF was introduced, its effects were examined, and the schema was broken down to remove violations. Following standard principles for database design, the final schema is now free of anomalies and redundancies.

5. Reference List

- Decomplexify. “Learn Database Normalization - 1NF, 2NF, 3NF, 4NF, 5NF.” *www.youtube.com*, 21 Nov. 2021, Available at: www.youtube.com/watch?v=GFQaEYEc8_8 [Accessed on December, 1st].

- GeeksforGeeks. “Enhanced ER Model.” *www.geeksforgeeks.org*, 24 Oct. 2017, Available at: www.geeksforgeeks.org/enhanced-er-model/ [Accessed on November, 25th].

- GeeksforGeeks. “Composite Attribute in DBMS.” *www.geeksforgeeks.org*, 10 Jun. 2024, Available at: www.geeksforgeeks.org/composite-attribute-in-dbms/ [Accessed on November, 26th].

- GeeksforGeeks. “Multivalued Attributes in DBMS” *www.geeksforgeeks.org*, 11 Jun. 2024, *Available at:* https://www.geeksforgeeks.org/multivalued-attributes-in-dbms/ [Accessed on November, 25th].

- Teesha Goyal. “Table Per Subclass” *www.naukri.com*, 27 Mar. 2024, Available at: https://www.naukri.com/code360/library/table-per-subclass [Accessed on November, 31st].

- W3Schools. “SQL FOREIGN KEY Constraint.” *www.w3schools.com*, 2019, Available at: www.w3schools.com/sql/sql_foreignkey.asp [Accessed on November, 28th].

- W3Schools. “SQL INSERT into Statement.” *www.w3schools.com*, 2019, Available at: www.w3schools.com/sql/sql_insert.asp [Accessed on November, 28th].

- W3Schools. “SQL DROP TABLE Statement.” *www.w3schools.com*, 2019, Available at: www.w3schools.com/sql/sql_drop_table.asp [Accessed on November, 28th].