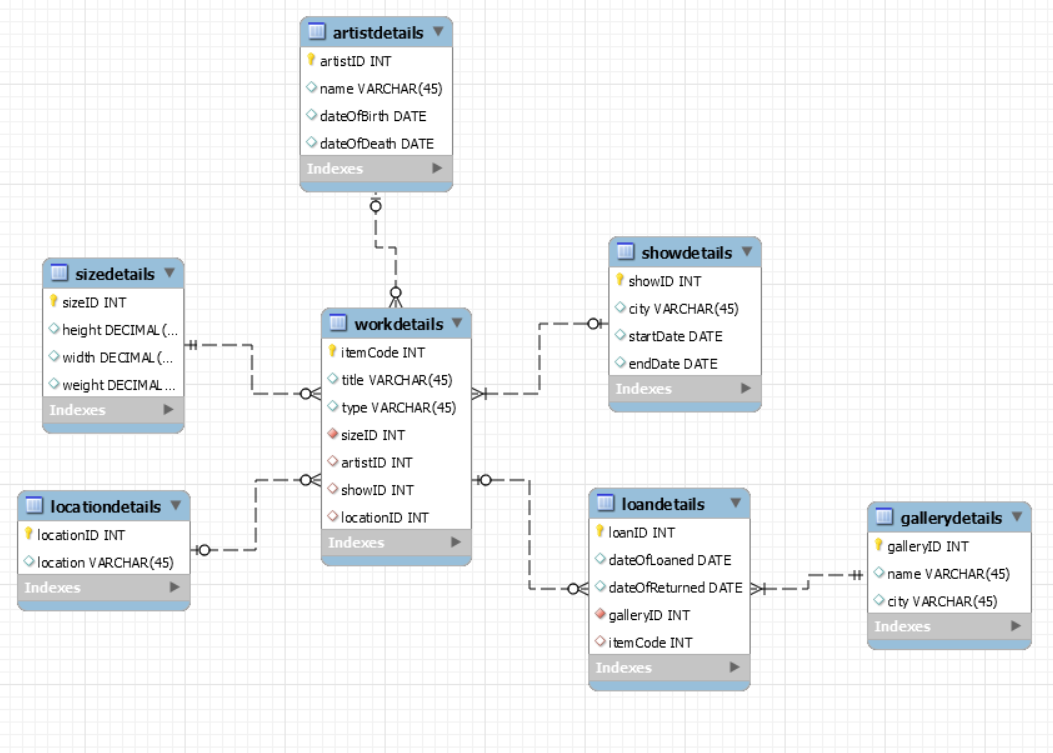
**Problem1:**

The following figure illustrates the entity relationship diagram of different works of art at museum (workdetails table) and their relationship with the following tables:



**Workdetails** : each work is identified by item code, title, type, and size

Primary key: itemCode

Foreign keys: sizeID, artistID, showID, LocationID

**sizedetails** : each size is composed of height, width and weight

Primary key: sizeID

**artistDetails** : work of art is developed by an artist and idenfied by artistID, name, date of birth and date of death

Primary key: artistID

**Locationdetails** : work of art is also described by its location within the museum

Primary key: locationID

**showdetails** : a work of art can be part of any travelling show and described by showed, city, start date and end date

Primary key: showID

**loandetails** : a work of art can be loaned to other galleries and identified by loanID, date of loaned, date of returned

Primary key: loanID

Foreign keys: galleryID

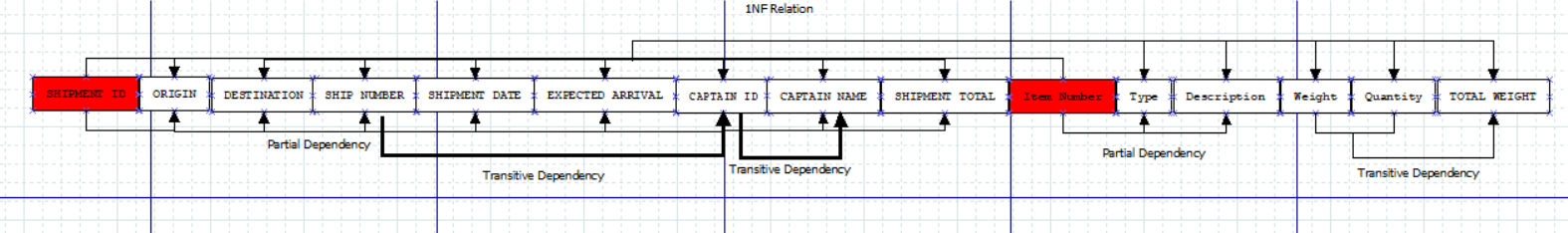
**gallerydetails** : a gallery is described by galleryID name and city

Primary key: galleryID

**Problem2**:

**a)Shipping Manifest**

**1NF Relation: (SHIPMENT ID, Item Number,** ORIGIN, DEPARTMENT, DESTINATION, SHIP NUMBER, SHIPMENT DATE, EXPECTED ARRIVAL, CAPTAIN ID, CAPTAIN NAME, SHIPMENT TOTAL, Type, Description, Weight, Quantity, TOTAL WEIGHT)



**Functional Dependency**:

(SHIPMENT ID, Item Number) -> (ORIGIN, DEPARTMENT, DESTINATION, SHIP NUMBER, SHIPMENT DATE, EXPECTED ARRIVAL, CAPTAIN ID, CAPTAIN NAME, SHIPMENT TOTAL, Type, Description, Weight, Quantity, TOTAL WEIGHT)

**Partial Dependency:**

(SHIPMENT ID) -> (ORIGIN, DEPARTMENT, DESTINATION, SHIP NUMBER, SHIPMENT DATE, EXPECTED ARRIVAL, CAPTAIN ID, CAPTAIN NAME, SHIPMENT TOTAL)

(Item Number) -> (Type, Description, Weight, Quantity, TOTAL WEIGHT)

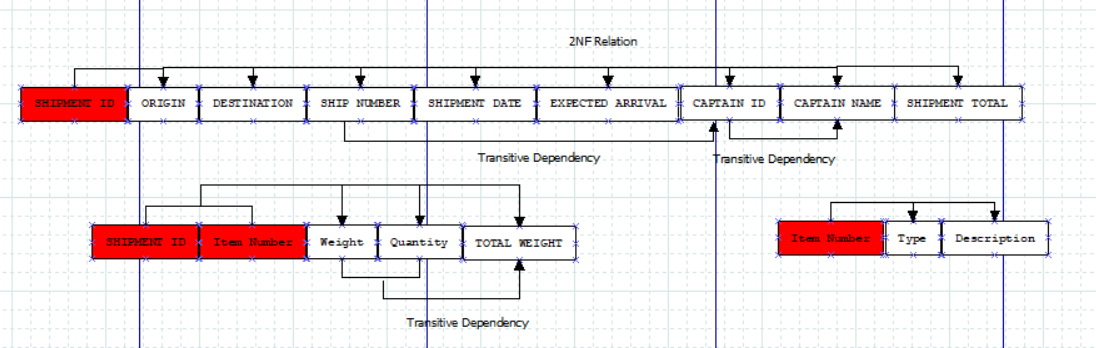
**Transitive Dependency:**

(SHIP NUMBER) -> (CAPTAIN ID)

(CAPTAIN ID) -> (CAPTAIN NAME)

(Weight, Quantity) -> (TOTAL WEIGHT)

**2NF Relation:**



**Table 1**

**Primary key: SHIPMENT ID**

**Foreign key: SHIPMENT ID (to Table 2)**

**Normal form: At least 2NF**

**Transitive dependency** (SHIP NUMBER) -> (CAPTAIN ID), (CAPTAIN ID) -> (CAPTAIN NAME)

**Table 2**

**Primary key: SHIPMENT ID + Item Number**

**Foreign key: Item Number (to Table 3)**

**Normal form: 2NF**

**Transitive dependency** (Weight, Quantity) -> (TOTAL WEIGHT)

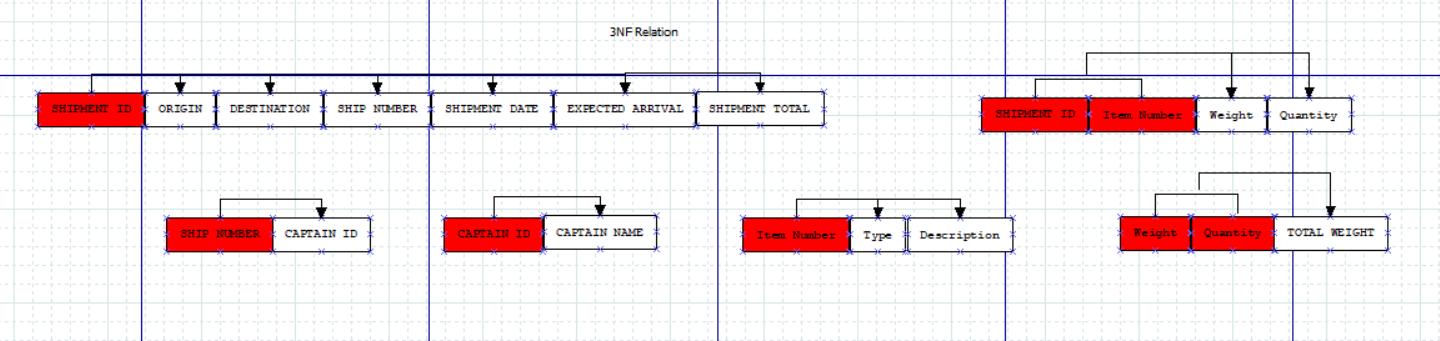
**Table 3**

**Primary key: Item Number**

**Foreign key: None**

**Normal form: 3NF**

**3NF Relation:**



**b)Parking ticket**

**1NF Relation**: **(Ticket#** , St\_ID, L\_Name, F\_Name, Phone\_No, St\_Lic, Lic\_No, Date, Code, Fine)

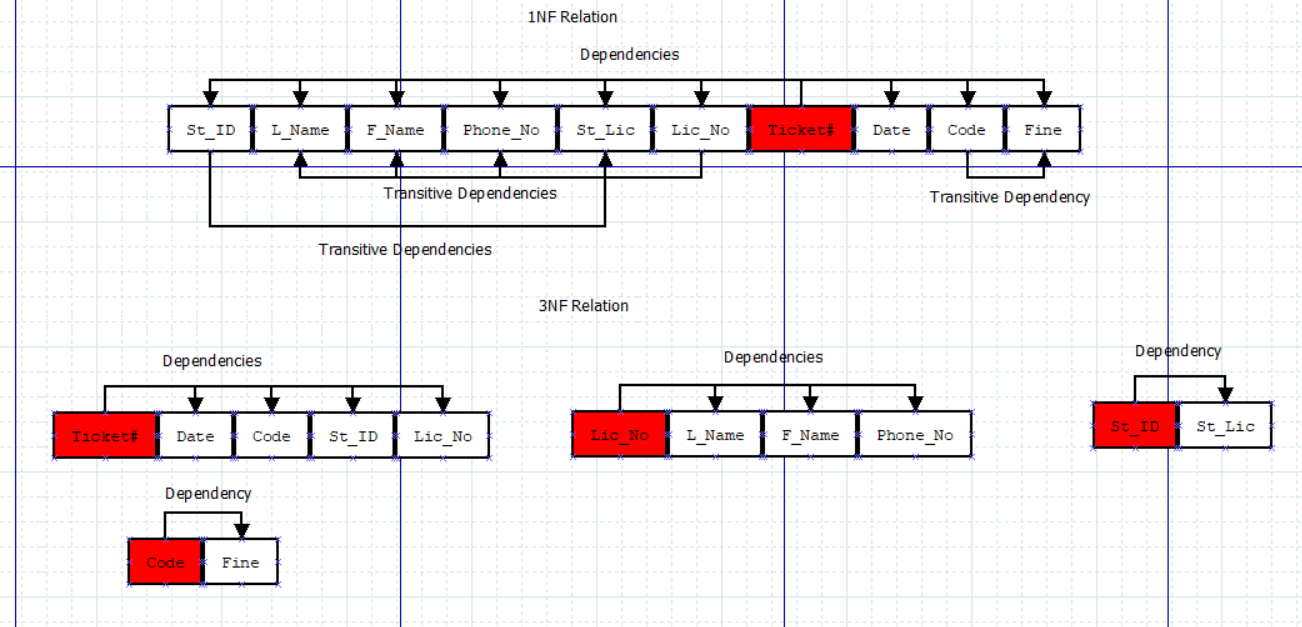
**Transitive Dependencies**:

(St\_ID) -> (St\_Lic)

(Lic\_No) -> (Phone\_No, F\_Name, L\_Name)

(Code) -> (Fine)

**3NF Relation**:



**Problem3**:

**Overview**:

This program helps to which invoices have been paid and which invoices have not been paid. A summary of the extraction requirements is in the csci3901 course assignment#6 information in the course’s Brightspace.

**Files and external data**:

The whole program is divided into different files to solve the program

Database\_Alter\_Schema.txt – contains the alter statements and update statements for mapping the orders and payments.

paymentManagement.java – Class that have three functions to reconcile the payments, pay for list of payments, to find the unknown payments and unpaid orders.

**Design Elements**:

**reconcilePayments**:

Operation: To connect orders to payments

Parameters: Connection

Description: To connect orders and payments and map the check numbers

Return Type: void

Algorithm:

* Iterated the result set of the payments table that are ordered by the payment date
* While iterating through the result set get the order details of the amount, check number and the customer and the orders which are shipped and resolved and are ordered by customer number and order date.

select \* from orders where status in ('Shipped', 'Resolved') and customerNumber = '"+ customerNumber +"' order by customerNumber, orderDate

* And iterated through the result set retrieved from the orders
* And will check for the check numbers that are not null
* Iterate through the orders and add to arraylist till it matches the order total to the amount payed.
* Format the calculated amount to the double value
* If found the respective match I will update the sql database.
* If not mapped the amount cleared the array list.

stmt3.executeUpdate("update orders \r\n" +

" set checkNumber = '"+ checkNumber +"' \r\n" +

" where customerNumber = '"+ customerNumber +"' and orderNumber = '"+ orderNumber +"';");

* If not mapped or found the check numbers un-map the other orders under the same customer number and proceed to the next check number.

**payOrder**:

Operation: To record the payment

Parameters: Connection, amount, cheque\_number, list of orders

Description: To record payments with the given cheque number for the list of orders

Return Type: boolean

Algorithm:

* Iterated the list of orders that are passed
* Retrieved the order total and customer number for the order where the check number is null

select orderTotal, customerNumber from orders where orderNumber = '"+ orderNumber+ "' and checkNumber is null;

* If the record is not populated for the given order return false
* If the customer number is not same for all the orders return false
* Calculate the total amount of all the orders and format to double value
* If the calculated total value is not equal to the amount payable, then return false
* Update all the orders with the given check numbers and return true

stmt3.executeUpdate("update orders\r\n" +

"set checkNumber = '"+ cheque\_number +"' \r\n" +

"where orderNumber = '"+ orderNumber +"';");

**unpaidOrders:**

Operation: to return the list of orders that are unpaid

Parameters: Connection

Description: To return the list of orders that are unpaid and no record of payment

Return Type: Arraylist

Algorithm:

* Execute the query for the orders that are not cancelled and disputed

select \*\r\n" +

" from orders \r\n" +

" where checkNumber is null and status not in ('Cancelled', 'Disputed');

* Added the orders to the arraylist
* Returned the arraylist.

**unknownpayments:**

Operation: to return the list of check numbers that are unknown

Parameters: Connection

Description: To record list of cheque number that are not payed

Return Type: Arraylist

Algorithm:

* Execute the query for the payments for the pairstatus that are null

"select \*\r\n" +

" from payments \r\n" +

" where pairStatus is null;"

* Added the payments to the arraylist
* Returned the arraylist.

**Assumptions:**

No past order was partially paid, and no past order was paid in installments to map the check numbers and return the unknown payments and orders.

**Constraints:**

**reconcilepayments method:** Orders that are shipped and resolved are considered

**unpiadOrders method:** Orders that are cancelled and disputed are not considered

**Why my code is ready for the live environment?**

* Database alterations for columns in orders and payments table are convenient to map the orders.
* Ability to adapt to new requirements and process such as orders that are not in process and on hold.
* Frequency of unnecessary updating and manipulating of database is very less.
* Several algorithms are written concise and changeable based on the requirement such as

1. Orders that are not payed chronologically
2. If the amount is not mapped to the check payed
3. If the orders are not present in the database

* Various test cases for all the methods are tested thoroughly